MÁSTER UNIVERSITARIO EN

INGENIERÍA INDUSTRIAL

TRABAJO FIN DE MÁSTER

PROGRAMACIÓN DE UN ROBOT COLABORATIVO DE DOS BRAZOS PARA RESOLVER EL CUBO DE RUBIK IMPLEMENTANDO VISIÓN ARTIFICIAL

Estudiante	*Herrán Angulo, Asier*
Director/Directora	*Cabanes Axpe, Itziar*
Departamento	Ingeniería de Sistemas y Automática
Curso académico	*2019/20*

Bilbao, 31, agosto, 2020

RESUMEN

El presente proyecto describe el diseño e implantación de un sistema formado por un robot colaborativo Yumi y una cámara de visión artificial, para la resolución de cubos de rubik de 3x3 y 2x2.

El proyecto abarca todas las fases del proceso. partiendo de la selección de la cámara hasta la implantación final, incluyendo el diseño de los elementos auxiliares necesarios y la programación y comunicación de los distintos componentes que toman parte en el proyecto.

El robot forma parte de una célula de montaje utilizada para la docencia en el departamento de Ingeniería de Sistemas y Automática de la Escuela de Ingeniería de Bilbao, y deberá de ser capaz de resolver el cubo de rubik sin interferir con los elementos que la componen, además de mantener un alto nivel de seguridad.

Una vez diseñada la distribución de la célula, se utilizará el programa RobotStudio para simular el proceso. De esta forma se comprobará que los movimientos y trayectorias programadas son correctos y que el robot es capaz de realizarlos.

A continuación, se implementará la estación real. Para ello, se realiza el diseño, impresión 3D y montaje de los elementos auxiliares necesarios, que permitirán colocar la cámara de visión artificial en posición.

Con la cámara fijada, se programará el procesamiento de imágenes mediante la herramienta Matlab. De esta forma se podrá obtener la información del cubo necesaria para resolverlo, es decir, el tipo de cubo con el que se trabaja, su posición en la mesa (ya que el usuario puede depositarlo aleatoriamente en cualquier posición) y la manera en la que está desecho (es decir, cómo de desordenados están los colores de las piezas que conforman las caras del cubo).

Partiendo de la información aportada por la cámara y siguiendo en Matlab, se programará un algoritmo que calcule los movimientos que resolverán cualquier configuración de los cubos de rubik de una forma eficiente.

Hecho esto, se establecerá una conexión TCP/IP entre el robot y Matlab mediante la cual se le enviarán los movimientos a realizar para la resolución del cubo.

Por último, se programará el robot real, implementando los movimientos programados mediante el simulador y validando el correcto funcionamiento de todo el sistema.

Palabras clave: Robótica colaborativa, visión artificial, comunicación TCP/IP, impresión 3D.

ABSTRACT

The following project describes the design and implementation of a system consisting of a Yumi collaborative robot and an artificial vision camera, for the resolution of both the 3x3 and 2x2 rubik´s cubes.

The project englobes all phases of the process, starting from the camera selection to the final implementation, including the design of the necessary auxiliary elements and the programming and communication of the different components that take part in the project.

The robot is part of an assembly cell used with the purpose of teaching in the department of systems and automation engineering of the Bilbao engineering school and should be able to solve the rubik´s cube without interference with the elements that compose it, in addition to maintaining a high level of security.

Once the cell distribution is designed, the RobotStudio program will be used to simulate the process. This way it will be verified that the movements and programmed paths are correct and that the robot is capable of carrying them out.

Hereunder, the real station will be implemented. For this, the design, 3D printing and assembly of the necessary auxiliary elements will be carried out, which will allow to place the artificial vision camera in position.

With the camera fixed in its place, the image processing will be programed using the Matlab tool. Thus obtaining information about the cube necessary to solve it, that is to say, the type of cube we are working with, its position on the table (since the user can deposit it randomly in any position) and the way that the cube its undone (the way the colors of the different pieces that make up the cube faces are positioned).

Starting from the information collected from the camera and following on Matlab, an algorithm will be programmed to calculate the movements that will solve any configuration of rubik´s cubes in an efficient way.

Done this, a TCP/IP connection will be established between the robot and Matlab through which the movements to be carried out for the resolution of the cube will be sent.

Finally, the real robot will be programmed, implementing the movements programmed through the simulator and validating the correct operation of the entire system.

Keywords: Collaborative robotics, artificial vision, TCP/IP connection, 3D printing.

LABURPENA

Proiektu honek Yumi robot kolaboratibo batek eta ikusmen artifizialeko kamera batek , 3x3 eta 2x2 rubik kuboak ebazteko osatutatzen duten sistema baten diseinua eta ezarpena deskribatzen du.

Proiektuak prozesuaren fase guztiak hartzen ditu, kamera eta robota aukeratzetik, azkenengo ezarpenera arte; beharrezko elementu osagarrien diseinua eta proiektuan parte hartzen duten osagaien programazioa eta komunikazioa barne.

Robota Bilboko Ingeniaritza Eskolako Sistemen eta Automatika Sailean irakaskuntzarako erabiltzen den muntaia-zelula baten parte da, eta rubik kuboa konpontzeko gai izan behar du, hura osatzen duten elementuekin interferentziarik sortu gabe, eta segurtasun-maila handia izan maintenduz.

Zelularen banaketa diseinatu ondoren, RobotStudio programa erabiliko da prozesua simulatzeko. Horrela, programatutako mugimendu eta ibilbideak zuzenak direla eta robota horiek egiteko gai dela egiaztatuko da.

Ondoren, estazio erreala ezarriko da. Horretarako, ikusmen artifizialeko kamera posizioan jartzeko beharrezkoak diren osagarrien 3D diseinua, inprimaketa eta muntaketa egingo da.

Kamera finkatuta, irudi-prozesamendua Matlab tresnaren bidez programatuko da. Hartara, kubo horri buruzko informazioa lortuko da, hau da, zer kubo-motarekin lan egiten den, zer posiziotan dagoen mahaian jarrita (erabiltzaileak edozein posiziotan utz baitezake ausaz) eta nola dagoen nahastuta (hau da, kuboaren aurpegiak osatzen dituzten piezen koloreak nola dauden kokatuta).

Kamerak emandako informaziotik abiatuta eta Matlab-en jarraituz, rubik kuboa edozein konfiguraziotik modu eraginkorrean ebatziko duen mugimenduak kalkulatzeko algoritmo bat programatuko da.

Hori egin ondoren, robotaren eta Matlab-en artean TCP/IP konexio bat ezarriko da, kuboa ebazteko egin beharreko mugimenduak bidaltzeko.

Azkenik, robot erreala programatuko da, simulagailuaren bidez programatutako mugimenduak inplementatuz eta sistema osoaren funtzionamendu egokia balidatuz.

Gako-hitzak: Elkarlaneko robotika, ikusmen artifiziala, TCP/IP komunikazioa, 3D inprimaketa.

Contenido

Tabla de ilustraciones

Índice de tablas

1. INTRODUCCIÓN

El presente proyecto se ha desarrollado en el Departamento de Ingeniería y Sistemas de la Escuela de Ingeniería de Bilbao, en la Universidad del País Vasco/Euskal Herriko Unibertsitatea. La robótica colaborativa está en auge y mediante este proyecto se quieren visibilizar las posibilidades y ventajas que aporta con respecto a la robótica tradicional.

Sin embargo, para entender el porqué de la importancia de la robótica colaborativa y de las demás tecnologías que toman parte en este proyecto, es necesario comenzar por describir el contexto en el que se encuentra.

El proyecto presentado se engloba dentro de la cuarta revolución industrial o Industria 4.0. En la actualidad, debido al gran desarrollo de las comunicaciones y medios de transporte, la oferta en el sector industrial se ha globalizado, provocando un aumento de la competencia y, por tanto, la necesidad en las empresas de mejorar la eficiencia de sus procesos de producción.

Por otro lado, los consumidores demandan productos cada vez más personalizados, por lo que flexibilizar las cadenas de producción se ha vuelto una necesidad para lograr seguir siendo competitivos en el mercado actual.

Estas condiciones han supuesto un cambio de paradigma en la industria, una nueva forma de entender el mercado y los procesos de producción que ha derivado en la cuarta revolución industrial. (Ilustración 1). [1]

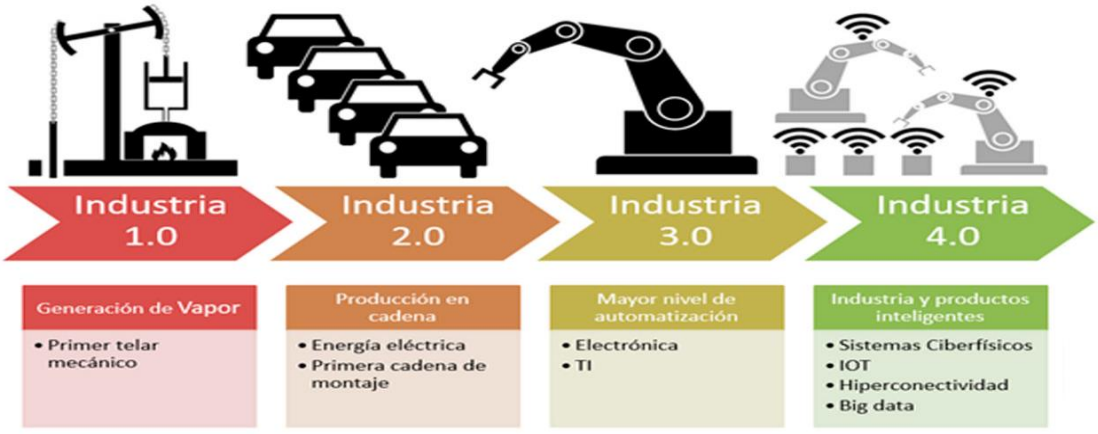


Ilustración 1. Esquema revoluciones industriales

A diferencia de las otras tres revoluciones industriales, es la primera vez que somos conscientes de que estamos viviendo una de ellas. Como consecuencia, numerosos países están desarrollando políticas e iniciativas que impulsen los desarrollos tecnológicos propios de esta revolución y permitan generar empresas cada vez más competitivas.

Dentro de estas iniciativas la más conocida es la Industrie 4.0 llevada a cabo por Alemania en 2013 con la intención de renovar su industria y crear las llamadas fábricas inteligentes, es decir,

fábricas con un alto grado de automatización, seguridad, gestión de datos y sostenibilidad que permitiesen a la industria alemana seguir a la vanguardia mundial. (Ilustración2). [2]

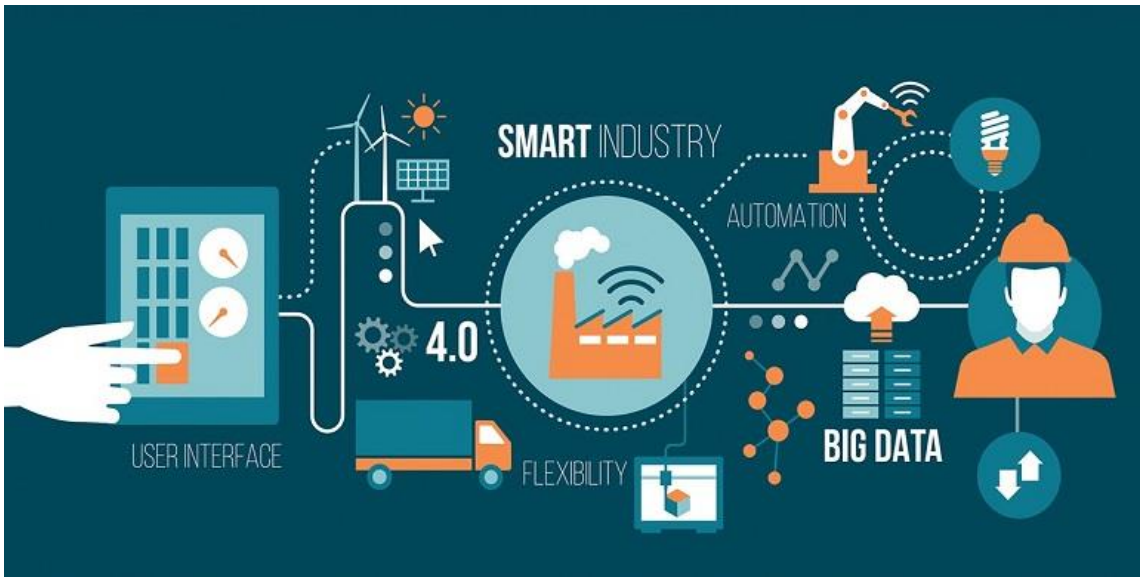


Ilustración 2. Fábricas inteligentes y características

A esta iniciativa se sumó el resto de Europa un año después con su proyecto Horizon 2020 y en 2021 renovando dicha apuesta con la iniciativa Horizon Europe, invirtiendo un total de 180 millones de euros en programas de investigación e innovación. [3]

Otros países como Estados Unidos, China o Japón han hecho lo propio con la creación de diversos programas cuyo fin no es otro que impulsar una producción más eficiente y ecológica.

Estas inversiones junto con la participación de millones de empresas y centros de investigación en todo el mundo están impulsando el desarrollo de nuevas tecnologías que permitan optimizar los procesos de producción. [4]

En el presente proyecto se trabajarán varias de estas tecnologías que están tomando una importancia clave en el avance de la Industria 4.0, como son los robots colaborativos, la fabricación aditiva o la simulación.

2. OBJETIVOS Y ALCANCE DEL TRABAJO

2.1. Objetivos

El objetivo principal del proyecto es el diseño e implantación de un sistema robótico capaz de interaccionar con las personas de manera segura para la resolución de los cubos de rubik de 3x3 y 2x2.

Este objetivo puede desglosarse en varios objetivos parciales:

- Diseño del sistema y la célula:

Como se ha comentado en apartados anteriores, el robot va incorporado en una célula de montaje utilizada con fines educativos. Por ello, es necesario diseñar un espacio de trabajo donde el robot pueda resolver el cubo cumpliendo los requisitos de seguridad y sin interferir con el resto de elementos de la célula.

- Modelado e impresión de componentes 3D:

Para adaptar la célula de ensamblaje al proyecto es necesario diseñar ciertos elementos auxiliares (carcasas, amarres, dedos de pinzas…) que permitan que el sistema funcione correctamente. Por tanto, en este objetivo se plantea diseñar dichos elementos, elaborar sus planos y realizar la impresión 3D de los mismos.

- Selección de la cámara y objetivo:

El robot por sí solo no es capaz de realizar el trabajo ya que necesita información del estado del cubo. Por tanto, uno de los objetivos es la selección de una cámara de visión artificial capaz de captar y compartir dicha información con el robot.

La cámara debe detectar los colores y cumplir con las especificaciones de resolución, además de tener una interfaz compatible con Matlab y una óptica que permita captar toda el área de trabajo del proyecto con una calidad aceptable.

- Aprendizaje de los programas RobotStudio y Matlab:

RobotStudio y Matlab son dos herramientas que permiten trabajar con el robot y con la cámara respectivamente, por lo que uno de los objetivos del proyecto es aprender a utilizarlos correctamente. De esta forma se podrán programar ambos componentes y sacarles el máximo partido, logrando que el conjunto funcione eficientemente.

- Programación del robot Yumi y de la cámara:

Tras adquirir los conocimientos necesarios de los programas con los que trabajan el robot y la cámara, el siguiente objetivo es programarlos.

En el caso del robot, la programación se hace en dos partes. En primer lugar, el software RobotStudio da la posibilidad de simular los movimientos y trayectorias que realizará el robot real, permitiendo comprobar si es capaz de realizarlos sin colisiones y de llegar a todos los puntos correctamente. Tras esto, el siguiente objetivo es implementar el código en el robot real, utilizando el FlexPendant para corregir los errores que hayan podido surgir por las diferencias entre la simulación y la realidad.

Respecto a la cámara, Matlab contiene las *toolboxes* necesarias para acceder a ella y realizar el procesamiento de imágenes. Por lo que, partiendo de ellas, se ha de programar la cámara para que tome y procese las imágenes del cubo, obteniendo así la información necesaria para resolverlo.

- Programación del algoritmo de resolución del cubo de rubik:

El objetivo final del proyecto es la resolución del cubo de rubik, por tanto, uno de los pasos más importantes es la programación de los algoritmos que resuelvan los cubos de 3x3 y 2x2. Estos algoritmos se realizarán en Matlab y serán enviados al robot mediante una conexión TCP/IP que también habrá de programarse.

- Implementación del sistema:

Una vez realizado lo anterior, el último objetivo es la implantación del sistema en la célula real, conectando todos los componentes y validando su adecuado funcionamiento, siendo capaz de resolver el cubo y cumpliendo con las medidas de seguridad pertinentes.

2.2 Alcance

En lo respectivo al alcance del proyecto, lo forman los siguientes hitos:

- Implementación y verificación del sistema en la célula de montaje.
- Simulación del sistema en RobotStudio.
- Diseño, impresión y elaboración de los planos de los elementos 3D.
- Redacción del documento correspondiente al Trabajo de Fin de Máster.

3. EVOLUCIÓN Y ESTADO ACTUAL DE LA TECNOLOGÍA

Desde la creación del primer robot programable en 1954, la robótica ha evolucionado hasta convertirse en parte indispensable de nuestras vidas. Hoy en día, existen robots en prácticamente cualquier ámbito de la sociedad, y cada día su número y sus capacidades aumenta, haciendo que sea inimaginable la vida sin ellos.

3.1 Industria 4.0

Históricamente, la robótica está muy ligada a la industria. De hecho, de los 2,5 millones de robots que se estima que hay en el planeta, el 90% se encuentra en células de montaje. La gran mayoría de los robots industriales llevan a cabo tareas repetitivas que a un humano le resultarían tediosas o peligrosas. Además, en muchos casos logran hacerlo con mayor precisión y eficiencia, ya que poseen mayor capacidad de carga y rango de movimientos.

Sin embargo, en la actualidad, los clientes solicitan productos cada vez más personalizados, por lo que las empresas para ser capaces de responder a dicha demanda se han visto en la necesidad de flexibilizar sus métodos de producción. Además, la globalización ha provocado un aumento en la oferta, lo que genera una gran competencia entre las empresas que las obliga a aumentar su eficiencia para ser competitivos en el marco actual. Esto ha supuesto un cambio de paradigma en la industria que ha derivado en la cuarta revolución industrial o Industria 4.0 [1].

La Industria 4.0 se basa en la creación de fábricas inteligentes que sean capaces de responder a las demandas de la sociedad de una forma rápida y eficaz. Para tal fin es indispensable la apuesta por los avances tecnológicos y por un uso eficiente de la información. Concretamente, la Industria 4.0 se sostiene sobre 9 pilares que se muestran en la ilustración 3 y se describen a continuación. [5,6]

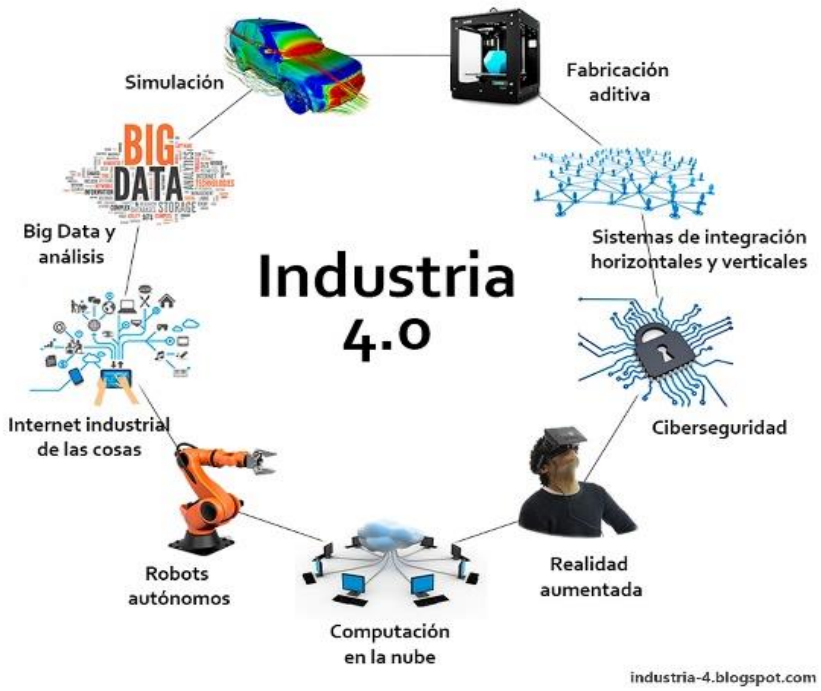


Ilustración 3. Pilares Industria 4.0

- **Big Data**: La gestión correcta de la información es una de las claves de la nueva revolución industrial. El Big Data consiste en el análisis y la gestión de grandes cantidades de datos, tanto provenientes de los propios procesos de la empresa como del entorno. Una gestión eficiente de la información aporta grandes beneficios a las empresas permitiéndoles optimizar procesos de producción, reduciendo consumos y mejorando la calidad de sus productos.

- **Simulación:** La generación de una copia de la maquinaria y células de producción en un entorno virtual (gemelo digital) posibilita la simulación del proceso productivo, permitiendo configurar, probar y optimizar los procesos antes de la implementación, lo que supone un ahorro de tiempo y dinero considerable a la hora de la puesta en marcha.

- **Fabricación aditiva:** También conocida como impresión 3D permite fabricar sin necesidad de utillajes piezas modeladas en entornos virtuales. De esta forma se pueden obtener piezas personalizadas a precios asequibles.

- **Sistemas de integración horizontal y vertical**: La cooperación y coordinación de todos los miembros de la empresa es crucial para lograr una producción eficiente. Por tanto, este pilar consiste en la creación de una única plataforma que interconecte los diferentes departamentos y facilite el trabajo conjunto de todos ellos.

- **Ciberseguridad:** Con el aumento de dispositivos conectados y de información compartida entre ellos, la ciberseguridad se ha convertido en una prioridad para evitar ataques que supongan grandes pérdidas.

- **Realidad aumentada:** La realidad aumentada aporta a los trabajadores información adicional en tiempo real, facilitándoles la toma de decisiones y las tareas a realizar.

- **Computación en la nube:** Mediante servidores distribuidos por todo el mundo, la nube permite intercambiar, compartir o en caso de ser necesario, recuperar datos de una forma sencilla a través de internet.

- **Internet de las cosas:** La comunicación en tiempo real entre humanos y máquinas es imprescindible para lograr un acceso rápido a la información. Mediante la implementación de sensores y dispositivos conectados a la red y capaces de comunicarse entre sí y con los operarios se logra facilitar la toma de decisiones individualizadas y optimizar los procesos de producción.

- **Robots autónomos:** La evolución de la robótica se dirige hacia robot cada vez más autónomos, capaces de recopilar información de su entorno y actuar en consecuencia, además de interactuar con otros robots o humanos compartiendo el mismo espacio de trabajo. En este contexto, los robots colaborativos están destinados a tener un papel clave.

3.2 Robótica colaborativa

Los robots colaborativos, o cobots, son robots industriales que comparten espacio de trabajo con humanos, permitiendo la interacción entre ellos o con otros robots. La colaboración entre humanos y robots remite grandes beneficios a la hora de realizar tareas repetitivas pero difíciles de automatizar, aumentando la eficiencia combinando la habilidad humana con la precisión y capacidad de trabajo del robot. [7]

Sin embargo, para Esben Østergaard, cofundador y director de operaciones de Universal Robots, la robótica colaborativa va más allá de que humano y robot compartan espacio de trabajo: «En nuestra opinión, ser colaborativos tiene mucho más que ver con ser accesibles y rebajar la barrera de la automatización poniendo nuestros robots al alcance de fabricantes que jamás pensaron que pudiesen llegar a plantearse instalarlos en sus centros de trabajo» [8]

Y es que, desde la instalación del primer robot colaborativo, el robot de Universal Robots UR5 (Ilustración 4), han sufrido un crecimiento imparable debido a que se amoldan perfectamente al paradigma actual, aportando flexibilidad y eficiencia a los procesos de producción.

Ilustración 4. Robot colaborativo UR5

El auge de este tipo de robots ha propiciado el desarrollo de numerosos estudios que comprueben su rendimiento, su impacto en la producción y los beneficios aportados en los procesos en los que se implementan.

Evidentemente, los resultados dependen de diversos factores como del tipo de sistema de producción o de las tareas que se encomienden al robot. Sin embargo, la mayoría de estudios analizados concuerdan en que la implementación de cobots permite optimizar los procesos de producción mejorando la producción y reduciendo costes. Así, en los estudios [9, 10, 11, 12] se han diseñado e implementado células de montaje híbridas donde robots y humanos trabajan juntos a fin de estudiar si realmente generan beneficios con respecto a una célula no automatizada. En todos casos los resultados fueron positivos, suponiendo un incremento de la productividad y de la calidad de trabajo de los operarios.

Como era de esperar tras los buenos resultados obtenidos, numerosas empresas han incorporado cobots a sus células de producción. En la empresa Aurolab, se ha instalado un robot UR5 de Universal Robots para facilitar el desarrollo de kits quirúrgicos para cataratas. El resultado fue tan satisfactorio que actualmente disponen de ocho robots colaborativos y su

producción ha aumentado un 15% [13]. Otro ejemplo es la empresa Stenner Pump que trabaja en el campo de la construcción. Desde la adquisición de sus dos primeros cobots son capaces de levantar paredes seis veces más rápido, además de reducir la intervención humana en procesos peligrosos como la recogida de ladrillos en un 75% [14].

Como se aprecia los ejemplos de robots colaborativos en la industria son cada vez mayores, no obstante, y aunque cada vez son capaces de realizar trabajos más complejos, hay ciertas tareas que no realizan con la misma eficiencia que las personas. Diversos estudios se han centrado en estudiar el comportamiento de robots y personas realizando diferentes tareas dentro de una cadena de montaje con el objetivo de discernir en qué situaciones instalar un robot puede aportar mayor beneficio.

En [15] se compara el desempeño de humanos y robots en tareas de distinta índole como apretar tornillos o trabajos de pick and place. Por otro lado, en [16] se han utilizado herramientas de simulación para analizar el rendimiento de robots y humanos en procesos de producción, teniendo en cuenta variables como la fatiga o la motivación de los trabajadores. Las conclusiones obtenidas son que los robots son mejores en tareas repetitivas debido a que no padecen cansancio, permitiéndoles mantener una precisión y rendimiento constantes. Por otro lado, los humanos desempeñan mejor tareas más complejas y que requieren una mayor inteligencia o destreza, y donde la fatiga no tiene tanta influencia (Ilustración 5).

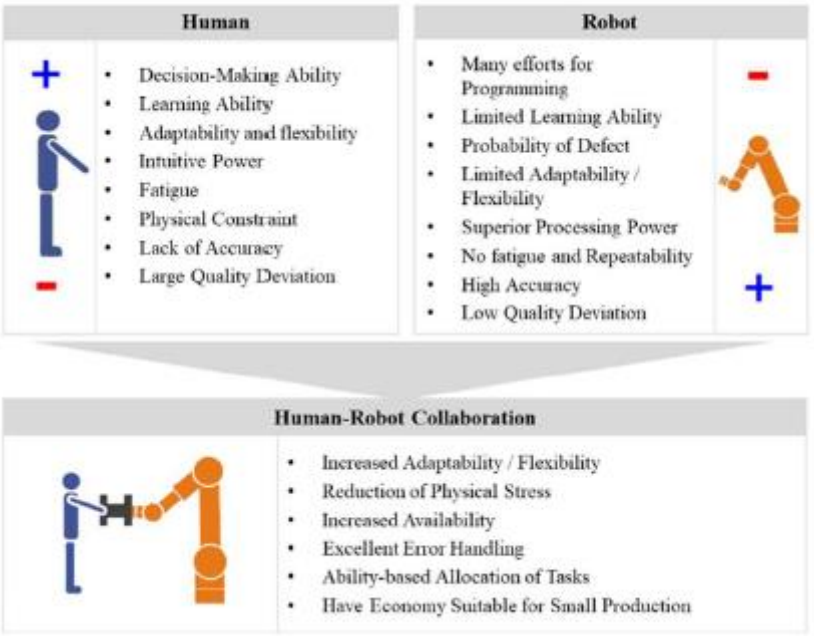


Ilustración 5. Características colaboración humano-robot

Su buen desempeño ha convertido a los cobots en un componente fundamental de la Industria 4.0, siendo el campo de la robótica que más ha crecido en la última década. Según el informe anual de Worlds Robotics [17], las ventas de robots colaborativos crecieron un 23% con respecto al año anterior, llegando hasta las 14000 unidades vendidas. Esta cifra supone únicamente el 3,27% del total de robots industriales vendidos, sin embargo, la tendencia de su crecimiento se mantiene al alza.

Por otro lado, en comparación con los robots industriales tradicionales, tienen un precio económico y son sencillos de instalar y programar, lo que combinado con su flexibilidad y la

mejora que generan en la productividad y calidad del trabajo, ha provocado su expansión a campos más allá de la industria.

Un claro ejemplo de ello es la medicina, donde desde que en 1983 se realizase la primera operación asistida por un robot, su número no ha dejado de crecer. Como se menciona en [18], en la actualidad existen más de 3000 robots Da Vinci en el mundo y participan en unas 200.000 intervenciones anuales. Como es de esperar, los cobots son también cada vez más habituales ya que gracias a su versatilidad y fácil implementación se utilizan para dar apoyo sanitario en tareas muy diversas.

En [19] se detalla la utilización de dos robots UR5 en el hospital universitario de Copenhague, en Gentofe a fin de optimizar el análisis y clasificación de las muestras de sangre. Su diseño interactivo y seguridad permiten a los operarios trabajar junto con el robot sin ningún tipo de protección adicional, logrando resultados realmente positivos, siendo capaces de analizar el 90% de las muestras en menos de una hora a pesar de que su volumen ha subido un 20% en el último año.

Por otro lado, empresas como Kuka que también han desarrollado robots colaborativos, están estudiando el uso de estos para ayudar en tareas de rehabilitación y de asistencia al movimiento para pacientes con movilidad reducida [20], entre otras aplicaciones.

En conclusión, los robots colaborativos son una realidad y su número y capacidades aumenta cada día, llevándoles a desarrollar tareas en todo tipo de ámbitos. Sin embargo, también la robótica se ha acercado al mundo del entretenimiento, donde se engloba este proyecto y donde la robótica tiene cada vez más protagonismo.

3.3 Robótica en el entretenimiento

Así, el ser humano ha tenido, desde hace miles de años, la idea de crear autómatas que les sirviesen y ayudasen en las tareas cotidianas, o que simplemente lograsen imitar acciones humanas. Ya en el siglo X a.C. un artesano llamado Yan Shi fabricó lo que diversos artículos consideran el primer autómata de la historia, un humanoide capaz de cantar y bailar. En los siglos posteriores le sucedieron otros muchos, cada vez capaces de realizar tareas más complejas, pero compartiendo la misión de entretener

En 1939 Joseph Barrett creó lo que es considerado como el primer robot humanoide de la historia, Elektro, The Robot Man (ilustración 6). Este humanoide de 2 metros y 120 kilogramos era capaz de realizar 26 movimientos diferentes y de reproducir 700 palabras. Ante la popularidad que adquirió el robot, el año siguiente desarrolló a Sparko, un perro robótico de 30 kilogramos capaz de ladrar, sentarse y de seguir varias órdenes sencillas.

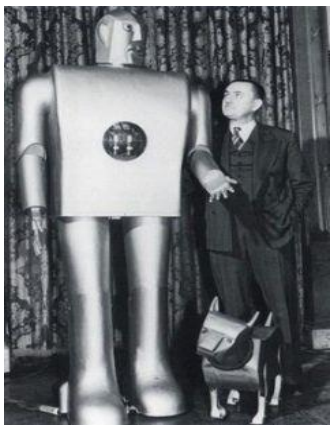


Ilustración 6. Elektro y Sparko

Hoy en día, gracias al gran desarrollo que han experimentado ciencias como la electrónica o la automática, existen en el mercado una gran cantidad de robots destinados al entretenimiento. Entre ellos, los más populares son las mascotas robóticas (Ilustración 7) las cuales poseen un gran rango de funcionalidades; desde las más simples que únicamente se mueven por la casa, hasta las más complejas, dotadas de inteligencia artificial y equipadas con cámaras que les permiten reconocer comandos de voz, esquivar obstáculos y hasta ayudar en las tareas del hogar.

Ilustración 7. Juguetes robot. Perro y humanoide

En el ámbito de los deportes se han llevado a cabo numerosos proyectos con el afán de crear robots deportistas. En [21] la empresa Kuka programó su robot Agilus para enfrentar al jugador profesional de ping pong Timo Boll como promoción para su nueva fábrica en china.

Otras empresas han optado por distintos deportes para promocionar sus robots, un ejemplo es Toyota [22] que en 2019 logró el record guiness de tiros libres anotados por un humanoide, logrando 2020 con su robot Cue3, en apoyo a la candidatura de Tokio para los juegos olímpicos de ese mismo año.

Los ejemplos de robots en el deporte son incontables, billar, hockey, incluso existen equipos de fútbol de robots, capaces no solo de jugar individualmente, si no entendiendo el deporte como un juego de equipo. [23]

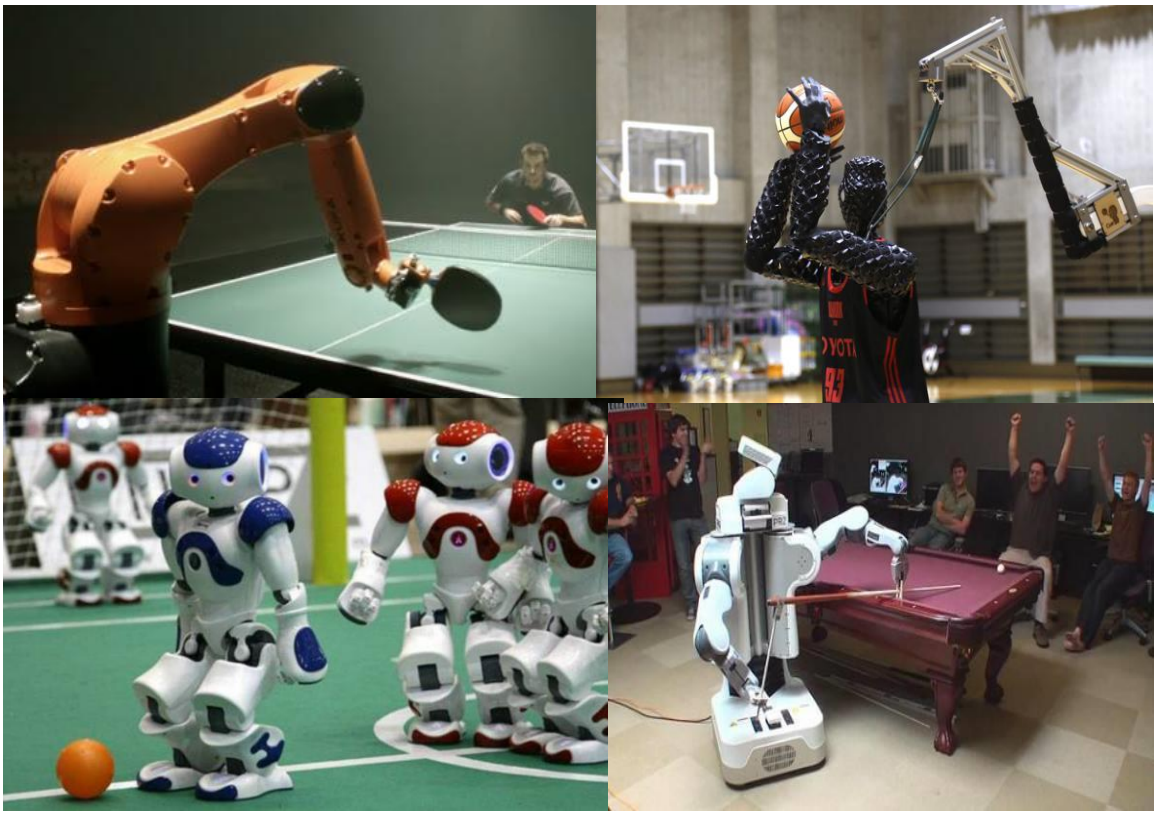


Ilustración 8. Robots practicando deporte

En el ámbito de las artes también comienzan a verse robots, llegando a abrir el debate si un robot es capaz de crear arte o no. Con el desarrollo de la inteligencia artificial, cada vez son más los robots capaces de pintar o componer sin seguir patrones fijos, únicamente basados en su aprendizaje.

Tal es el éxito de alguno de estos robots que incluso han presentado exposiciones de arte y vendido cuadros por cientos de miles de dólares. El caso más conocido es AI-DA un robot humanoide dotado con inteligencia artificial que le permite crear cuadros basados en su propio aprendizaje, y que actualmente tiene una exposición de Oxford. [24]

En el ámbito de la música existen también múltiples robots capaces de tocar instrumentos, algunos simplemente reproduciendo piezas ya creadas y otros improvisando (Ilustración 9). Sin embargo, cabe destacar a la cantante Hatsume Miku que, a pesar de no ser un robot, ya que es

un holograma, es sin duda el caso más famoso de una inteligencia artificial abriéndose paso en el mundo de la música interpretando canciones creadas por inteligencias artificiales.

Ilustración 9. Robots pintando y tocando música

Por último, un campo que está experimentando un gran crecimiento es el de los robots de asistencia para personas mayores o enfermas. Estos robots no solo sirven para entretener, (siendo capaces de jugar a juegos, poner música o hablar), sino que también ayudan en las tareas del hogar, o a llevar un control de las constantes vitales y, en definitiva, a asistir al usuario en sus necesidades.

Como se analiza en [25], los cuidados de personas mayores o enfermos siempre han estado ligados a cuidadores o a personal sanitario. Y es que se tratan de tareas tan delicadas y complejas que es realmente difícil que un robot pueda realizarlas autónomamente. Sin embargo, en los últimos años han surgido numerosos proyectos para facilitar las tareas de los cuidadores mediante la asistencia de robots.

En [26] se recoge el desarrollo de un robot interactivo para ayudar a niños con autismo a mejorar sus habilidades sociales y de comunicación. El robot puede adecuarse a cada niño ¿y conjunto con un grupo de expertos?? se han implementado diversos juegos y actividades que les permitan desarrollar su potencial en las primeras fases del trastorno.

Por otro lado, en [27] se presenta la creación de un robot para ayudar a las personas con demencia en su día a día. Este robot es capaz de reconocer rostros y comunicarse con las personas además de contar con varias actividades que les ayuden a sobrellevar la enfermedad.

3.4 Visión artificial

Uno de los factores claves en el avance de la automatización es la evolución que han sufrido los sistemas de visión artificial en los últimos años. La visión artificial es la tecnología que engloba los sistemas capaces de capturar y procesar datos a través de imágenes.

El primer sistema de visión artificial fue desarrollado por Larry Roberts en 1961, cuando creó un sistema capaz de percibir figuras en tres dimensiones, procesarlas y plasmarlas en dos dimensiones, sentando las bases de la visión artificial. (Ilustración 10).

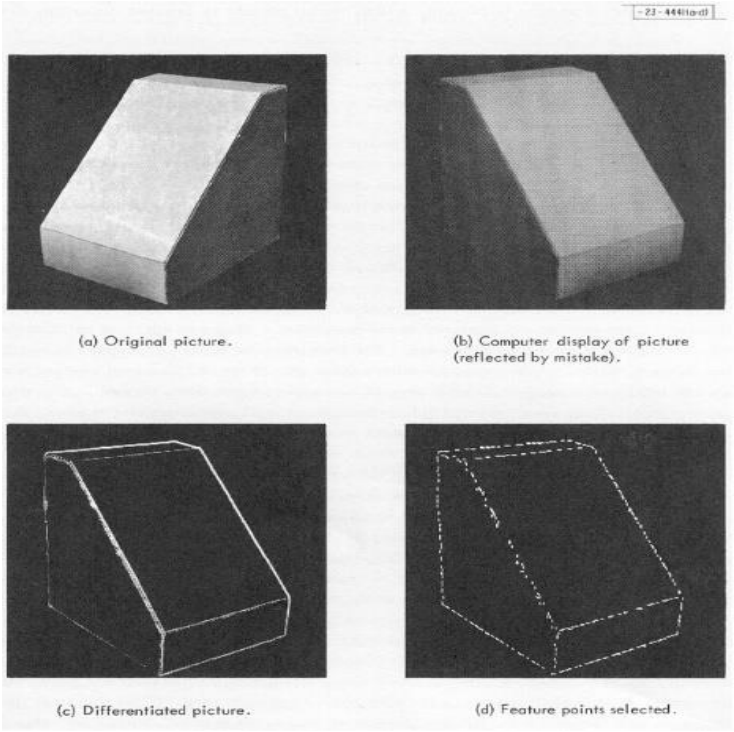


Ilustración 10. Proceso del primer procesamiento de imágenes

A partir de ese momento, se crearon infinidad de sistemas con cada vez mayor capacidad, pero no fue hasta la década de 1980 con la nueva generación de microprocesadores, cuando se comenzaron a utilizar ampliamente en la industria.

En la actualidad, los sistemas de visión artificial son un componente fundamental en los procesos de producción, permitiendo automatizar procesos como detección de defectos, metrología, seguimiento o procesos de bin picking, entre muchos otros.

Además, gracias a la evolución de otras tecnologías como la inteligencia artificial o el almacenamiento de datos, los sistemas de visión son cada vez más sofisticados y potentes, capaces de reconocer objetos más complejos en periodos de tiempo muy breves y con índices de acierto altísimos. Como consecuencia, y pese a que el campo donde su uso predomina es la industria, los sistemas de visión se han implantado en ámbitos de todo tipo como la medicina o el entretenimiento, entre otros.

En el sector industrial son muchísimos los ejemplos de aplicaciones que hacen uso de la visión artificial. Por ejemplo, en [28] se ha desarrollado un sistema de visión ultravioleta para la detección de defectos en frutas. Este sistema ha sido dotado de inteligencia artificial, que una

vez entrenada ha logrado una precisión del 96.5% en su cometido, lo que permite una clasificación mucho más rápida y eficiente que la realizada manualmente hasta la fecha.

Otro de los usos más comunes de este tipo de sistemas es el bin picking que consiste en que el robot sea capaz de, en un entorno desordenado, encontrar y coger un objeto concreto para posteriormente colocarlo en el lugar correcto. Uno de los últimos estudios realizados al respecto [29] detalla el desarrollo de un sistema de visión dotado con inteligencia artificial capaz de reconocer la pieza deseada incluso aunque esté parcialmente tapada por otras.

Por otro lado, los sistemas de visión artificial son indispensables en el ámbito de la robótica en el entretenimiento. Los robots deben de ser capaces de responder a estímulos externos como pueden ser una pelota acercándose o un gesto de un humano con el que interactuar. Para ello llevan incorporados cámaras y procesadores capaces de tomar y analizar cientos de imágenes por minuto, lo que les permite disponer en todo momento de la información necesaria de su entorno y actuar en consecuencia.

Ejemplo de este tipo de robots son varios de los mencionados en el apartado anterior, sin embargo, cabe destacar a Forpheus (Ilustración 11), un robot desarrollado por la empresa Omron dotado con inteligencia artificial, sensores de movimiento y cámaras de gran velocidad capaz de no solo jugar a ping pong, si no de adaptarse a las capacidades de su oponente, darle consejos e incluso aprender de él para mejorar [30].

Ilustración 11. Robot Forpheus

4. ANÁLISIS DE ALTERNATIVAS

En este apartado se describen las diferentes alternativas valoradas para la realización del proyecto, y se justifican en cada subapartado las elecciones tomadas para la solución final.

4.1. Selección del robot

El robot es el componente central del sistema y el de más valor económico, por lo que su correcta selección es un punto crítico en el proyecto.

En primer lugar, debido a que el proyecto se basa en la interacción del robot con el usuario y a que se lleva a cabo en un entorno docente donde se utilizará para futuras prácticas con los alumnos, se ha optado por un robot colaborativo frente a uno industrial. Aparte de un precio más económico, su facilidad a la hora de programar y su flexibilidad, añadido a la posibilidad de interactuar con ellos lo hacen la mejor opción.

Por otro lado, dentro del mercado de los robots colaborativos se ha optado por uno de dos brazos, ya que facilitará enormemente la resolución del cubo de rubik.

Llegado a este punto, las alternativas son el robot Yumi – IRB1400 de la empresa ABB y el Hiro Nextage de la impresa Kawada Industries.

- **Robot Yumi IRB 14000:**

El robot Yumi es un robot colaborativo de dos brazos desarrollado por la empresa ABB en 2015. Su diseño (recubrimiento acolchado) y sistemas de seguridad integrados lo hacen perfecto para el trabajo conjunto y la interacción con otros humanos o robots.

Sus dos brazos, de siete grados de libertad cada uno, le aportan un gran rango de movimientos a lo largo de su espacio de trabajo, permitiéndole manipular cargas de 0.5 kg y realizar movimientos con una repetitividad de 0.3 mm.

Además, el Yumi posee dos pinzas colaborativas Smart Gripper que facilitan la manipulación de piezas de pequeño tamaño. Estas pinzas incorporan una cámara de visión artificial y ventosas que facilitan las tareas de localización y movimiento de piezas.

Por otro lado, su controladora, permite la conexión con componentes externos de una manera sencilla, pudiéndose así generar sistemas más complejos y con mayor capacidad.

- **Robot Hiro/Nextage:**

El robot Hiro fue desarrollado en 2009 por la empresa japonesa Kawada Industries como su apuesta en el mercado de los robots colaborativos.

Hiro cuenta con una plataforma móvil y dos brazos robóticos de seis grados de libertad cada uno, que sumado al de su cintura, le aportan un espacio de trabajo enorme.

Además, cuenta con cuatro cámaras de visión artificial, dos en la cabeza y una en cada brazo, convirtiéndole en un sistema realmente completo y versátil.

Por último, sus motores de baja potencia (80W) gobernados por un robusto sistema de seguridad le permiten la interacción con humanos con un nivel de seguridad elevado.

Tras analizar ambos robots se llega a la conclusión que ambos son válidos para el proyecto, por lo que en la tabla 1 se realiza una comparación de sus características más importantes para facilitar la toma de la decisión.

Robot	GDL	Repetitividad (mm)	Carga por brazo (Kg)	Precio (€)	Imagen
Yumi	14 7 por brazo	0.02	0.5	35.000	
Hiro	15 2 en el cuello 1 en la cintura 6 por brazo	0.03	1.5	50.000	

Tabla 1. Comparativa entre Yumi y Hiro

Como puede observarse en la tabla 1, Hiro cuenta con un grado de libertad más y una mayor capacidad de carga, mientras que el Yumi tiene una mejor repetitividad y un precio más asequible.

Con respecto a los grados de libertad, para la resolución del cubo de rubik los movimientos de la cadera o del cuello del Hiro apenas aportan beneficio alguno. Por otro lado, tras analizar sus espacios de trabajo, ambos robots son capaces de realizar sin problema los movimientos necesarios para resolver el cubo, por lo que en este apartado no hay una gran diferencia entre ambos.

Además, pese a que la capacidad de carga del Hiro es evidentemente superior, el peso combinado del cubo y las pinzas apenas alcanza los 0.4 kg, por lo que el Yumi es capaz de manipularlo correctamente.

Por último, uno de los objetivos del proyecto es la implementación de un sistema de visión artificial que sea capaz de trabajar junto con un robot colaborativo. El Hiro ya lo incorpora per se, lo que restaría valor y capacidad de aprendizaje a dicho objetivo.

En conclusión, y teniendo en cuenta que ambos robots son válidos para el proyecto, la diferencia de precio ligada a la experiencia y buen hacer de la empresa ABB decantan la balanza favor del Yumi.

4.2. Selección de la cámara de visión artificial

Dejando de lado al robot, la cámara es el componente más importante del proyecto ya que es imprescindible para toma de imágenes que aporten la información del entorno que permita resolver el cubo.

En el mercado actual hay una gran cantidad de cámaras de visión artificial por lo que en primer lugar se van a describir las especificaciones que ha de tener la seleccionada para el proyecto.

Debido a que su principal cometido es el de tomar fotos del cubo de rubik para su posterior procesamiento, la cámara ha de ser capaz de tomar fotografías a color.

Por otro lado, para facilitar el posterior procesamiento de las imágenes, la resolución es una característica a tener en cuenta. Evidentemente, una mayor resolución repercute en el precio, por lo que se ha estimado que la cámara debe de tener una resolución mínima de 2K para que el procesamiento de las imágenes (especialmente de la primera) sea efectivo.

Por último, el robot Yumi seleccionado en el apartado anterior incorpora la posibilidad de conectar cámaras externas por medio de uno de sus puertos RJ45 situado en la controladora. Para aprovechar esta circunstancia, se ha optado por una cámara con una interfaz gigE (Gigabit Ethernet).

Como es evidente, existen una gran cantidad de fabricantes que ofrecen cámaras que cumplen estas características, sin embargo, debido a la buena relación existente con la empresa Infaimon se ha optado por elegir entre su catálogo.

Cabe destacar que la cámara va a ser utilizada para obtener información del cubo de rubik, y posteriormente para proyectos de carácter docente. Es decir, no es necesario que sus prestaciones sean tan elevadas como las de cámaras de cuyo funcionamiento dependa la seguridad el sistema o los usuarios. Por tanto, se va a optar por una cámara que cumpla las especificaciones mínimas al precio más asequible.

Teniendo en cuenta lo comentado hasta ahora, se ha optado por la familia de cámaras UI 5584LE. Estas cámaras, aparte de cumplir con las necesidades mencionadas aportan una gran flexibilidad tanto a la hora de la alimentación como en el intercambio de datos. Esto se debe a que, además del puerto RJ45 y del de alimentación de dos polos, cuentan con un puerto ZIF de 10 polos que permite la conexión con placas externas de todo tipo.

Por último, entre las cámaras que forman parte de esta familia se ha escogido la UI 5584LE-C-HQ. Esta cámara tiene la peculiaridad de tener una gran sensibilidad lumínica, ya que utiliza la tecnología de píxeles A-Pix que aporta una gran precisión y nitidez en los colores; lo cual supone una gran ventaja en un proyecto donde la distinción de colores en tan importante como es la resolución de un cubo de rubik.

Ilustración 12. Cámara UI 5584LE-C-HQ

4.3. Selección de la configuración del sistema

Tras la selección de la cámara y el robot, componentes principales del sistema, es necesario decidir cómo conectarlos entre sí para sacarles el máximo partido. Para este problema existen dos posibles configuraciones:

- **Conexión de la cámara directamente al robot:**

Como se ha explicado en el subapartado anterior, la cámara seleccionada cuenta con una interfaz gigE que le permite conectarse al robot mediante una conexión Ethernet IP. Una vez conectada, RobotStudio aporta la posibilidad de acceder a la cámara y programarla desde el propio programa. De esta forma una vez se cargue el código desde el RoboStudio al robot, no sería necesario ningún ordenador para que el sistema funcione.

- **Conexión de la cámara y el robot a través de un ordenador:**

Otra de las posibilidades es utilizar un ordenador como nexo entre la cámara y el robot. Esta configuración obligaría a utilizar un programa capaz de comunicarse y de intercambiar los datos captados por la cámara con el robot. Sin embargo, por otro lado, aporta una mayor flexibilidad al sistema para la programación de la cámara y el procesamiento de imágenes.

Respecto al procesamiento de imágenes, RobotStudio es lo suficientemente potente como para obtener la información necesaria para el proyecto, sin embargo, es poco versátil en comparación con otros programas orientados específicamente a la visión artificial o con herramientas como Python o Matlab que permiten crear tus propias aplicaciones para el procesamiento de imágenes.

Asimismo, la no utilización de un ordenador implica la obligatoriedad de programar el algoritmo de resolución del cubo de rubik en RobotStudio, y, por ende, utilizar el lenguaje Rapid, claramente orientado a la programación de robots. La generación del algoritmo es posible, no obstante, además de ser una programación más tediosa que con otros lenguajes más versátiles, no permite generar un algoritmo tan complejo y eficiente, lo que deriva en un mayor número de movimientos necesarios para resolver el cubo.

En conclusión, se ha optado por utilizar un ordenador como puente de conexión entre la cámara y el robot. La flexibilidad que aporta a la hora de programar la cámara y de generar el algoritmo para la resolución del cubo permiten crear un sistema mucho más eficiente. Además, de cara a la docencia y a futuros proyectos, el ordenador aporta un mayor abanico de posibilidades, por lo que se selecciona esta opción para el proyecto. (Ilustración 13).

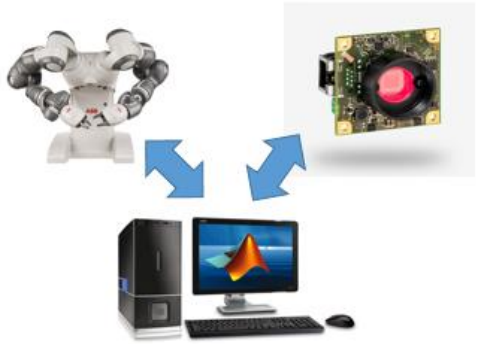


Ilustración 13. Esquema conexión elementos principales del sistema

4.4. Selección del software para la programación del algoritmo de resolución del cubo de rubik y el procesamiento de imágenes.

Al seleccionar el ordenador como nexo entre la cámara y el robot, se abre un abanico posibilidades a la hora de realizar tanto la programación del algoritmo de resolución del cubo de rubik como la de la cámara y el procesamiento de imágenes.

La primera de las opciones es la utilización del RobotStudio, herramienta de programación propietaria que el fabricante del robot aporta con el robot Yumi de ABB seleccionado en el apartado 4.1. Sin embargo, como se ha comentado anteriormente, es la opción más compleja y que peores resultados obtiene a la hora de resolver el cubo. Ello se debe a la falta de comandos para optimizar el algoritmo inteligente que resuelva el cubo de rubik, así como de los comandos para el procesamiento de imágenes. Por ello, y por la falta de flexibilidad que aporta de cara a futuros proyectos, esta alternativa queda descartada.

La otra posibilidad es la utilización de lenguajes de programación de propósito general como Python o Matlab.

Python es uno de los lenguajes de programación más populares en la actualidad gracias a su apuesta por la simplicidad, rapidez y versatilidad. Entre sus ventajas destaca que está preparado para realizar cualquier tipo de programa, desde aplicaciones de Windows hasta páginas web y que es compatible con una gran variedad de plataformas como Mac, Unix, Windows, etc. Además, es un lenguaje interactivo e interpretado, lo que junto a la gran cantidad de funciones y librerías que incorpora permite programar de una forma rápida y sencilla.

Por otro lado, Matlab es uno de los programas computacionales más extendidos, con una gran capacidad para manejar y resolver cálculos matemáticos. Aunque inicialmente fue diseñado para trabajar con matrices y vectores (su nombre proviene de “MATrix LABoratory”), su evolución ha sido enorme, y hoy en día es uno de los programas más utilizados para la resolución de todo tipo de problemas relacionados con la ingeniería y la ciencia en general. Al igual que Python, es un lenguaje interactivo y compatible con una gran cantidad de software como Excel, C++ o Fortran. Asimismo, posee una gran cantidad de toolboxes orientadas a la resolución de problemas de ingeniería avanzados como la inteligencia artificial o el procesamiento de imágenes o videos entre otros, y da la posibilidad de elaborar programas y aplicaciones ejecutables en cualquier dispositivo.

Cabe destacar que ambos lenguajes permiten realizar tanto la programación del algoritmo de resolución del cubo de rubik como del procesamiento de imágenes de una forma sencilla y eficaz, por lo que, por simplicidad del sistema, se utilizará el mismo lenguaje para la generación de ambos programas.

Python es el lenguaje que mayor crecimiento está sufriendo en los últimos años, sustituyendo al propio Matlab en muchos ámbitos. Sin embargo, para este proyecto, la capacidad matemática de Matlab y su desarrollo en base a matrices hace que la programación del algoritmo de resolución del cubo de rubik sea más sencilla. Esto se debe a que cada una de las caras del cubo se sustituye por una matriz de 3x3 para facilitar su resolución, por tanto, trabajar con un lenguaje orientado a matrices, permite disponer de una gran cantidad de funciones que optimicen el código, logrando un algoritmo más eficiente y reduciendo así el número de movimientos.

Por otro lado, la gran cantidad de toolboxes dirigidas hacia la programación de cámaras y al procesamiento de imágenes, desarrolladas tanto por el propio Mathworks como por otros usuarios de Matlab, simplifica mucho la programación de dichos programas, convirtiendo a Matlab en la mejor opción para este proyecto.

5. DESCRIPCIÓN DE LA SOLUCIÓN

En este apartado se describe el proceso seguido para la realización del proyecto. Como se ha comentado anteriormente, el objetivo final es la resolución del cubo de rubik mediante un robot colaborativo de dos brazos, para lo cual se implantará un sistema formado por el robot Yumi y una cámara de visión artificial. En cuanto al software, la solución que se plantea se divide en dos grandes bloques. Por un lado, la herramienta Matlab, mediante la cual se realiza la toma de imágenes, su procesamiento y el cálculo de los movimientos para resolver el cubo de rubik. Y, por otro lado, el robot Yumi que será en encargado de realizar la resolución del cubo.

5.1 Elementos principales del sistema

El sistema está formado tres componentes principales: un robot colaborativo Yumi, una cámara de visión artificial UI 5584 LE-C-HQ y un ordenador de mesa que hace de nexo entre ambos mediante la herramienta Matlab. (Ilustración 14).

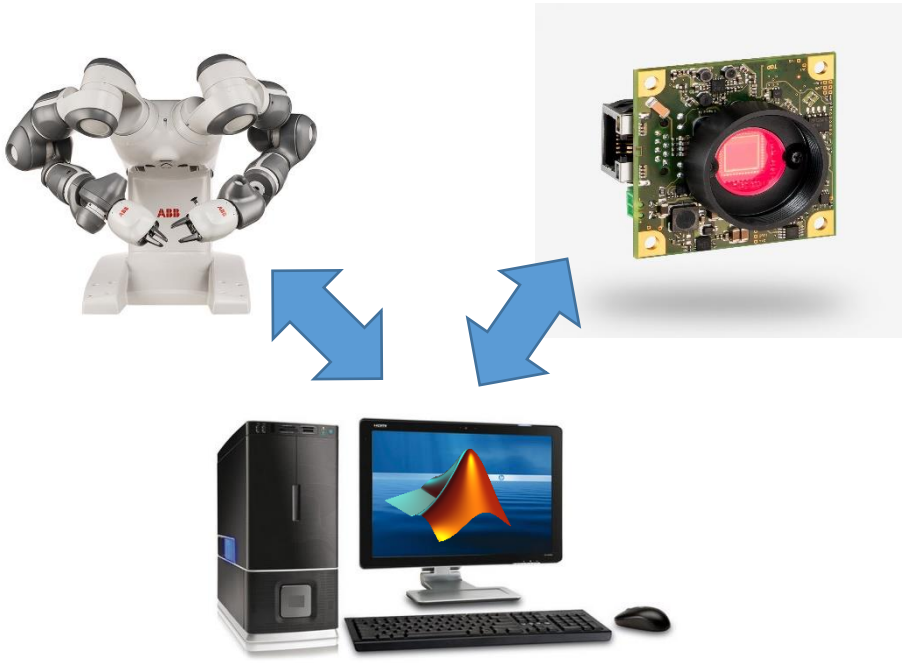


Ilustración 14. Esquema componentes sistema

Para el correcto funcionamiento del sistema es fundamental la sincronización y comunicación entre los tres elementos, ya que al igual que cada componente es indispensable por las tareas que realiza, la cooperación entre ellos también lo es.

La cámara se encarga de tomar las fotos que proporcionen al robot la información correspondiente al cubo. En primer lugar, se toma una foto de la mesa de trabajo de donde se extrae el tipo de cubo a resolver y su posición. Posteriormente, toma una foto a cada cara del cubo que permita conocer como está mezclado.

El robot por su parte, es el encargado de manipular e interaccionar con el cubo, es decir, de cogerlo de la mesa, llevarlo a las posiciones programadas para la toma de imágenes y de resolverlo en última instancia.

Por último, Matlab permite la sincronización de ambos durante la toma de imágenes y realiza el procesamiento de las imágenes, obteniendo la información necesaria en cada caso. Además, partiendo de dicha información, calcula los movimientos necesarios para resolver el cubo y se los envía al robot para su ejecución.

Dicho esto, y a modo de introducción a la descripción de la solución, se muestra una lista de los pasos dados para la resolución del cubo:

(1) Sincronización de los elementos del sistema
    a. Configuración de la comunicación TCP/IP entre Matlab y el robot
    b. Configuración de la comunicación entre Matlab y la cámara

(2) Toma del cubo de la mesa, independientemente de su posición
    a. Sincronización con la cámara para la toma de la imagen de la mesa
    b. Cálculo del tipo de cubo y de su posición a partir de la imagen
    c. Envío de la información al robot para ejecutar los movimientos
    d. Ejecución de los movimientos por parte del robot para tomar el cubo

(3) Toma de las fotos de cada una de las caras del cubo
    a. Sincronización entre el robot y la cámara para tomar las imágenes en el momento correcto
    b. Generación de las trayectorias sincronizadas entre ambos brazos del robot para la manipulación del cubo

(4) Cálculo de los movimientos a realizar para la resolución del cubo
    a. Obtención del estado del cubo partiendo de las imágenes de las caras
    b. Envío de los movimientos al algoritmo de resolución del cubo
    c. Envío de los movimientos calculados al robot

(5) Resolución del cubo de rubik
    a. Generación de las trayectorias óptimas para realizar los movimientos que resuelvan el cubo

5.2. Diseño de la zona de trabajo

El robot Yumi se encuentra incorporado en una célula de montaje realizada por dos estudiantes durante el curso académico 2018-2019. Esta célula se utiliza con fines educativos en los másteres de Ingeniería en Tecnología Industrial y de Automatización y Robótica, por lo que es imprescindible que esta nueva aplicación no interfiera con dichos escenarios.

5.2.1 Mesa de recepción del cubo de rubik

Como se aprecia en la ilustración 15, el robot está acompañado de una mesa en la que hay varios dispensadores de piezas y zonas de almacenaje que facilitan las tareas de ensamblado. Para la resolución del cubo, aparte de la cámara, es necesario añadir una superficie plana donde el usuario depositará de forma aleatoria el cubo. A partir de ahí, el robot localizará de forma autónoma el cubo, lo cogerá y lo resolverá, depositándolo a la superficie plana una vez resuelto.

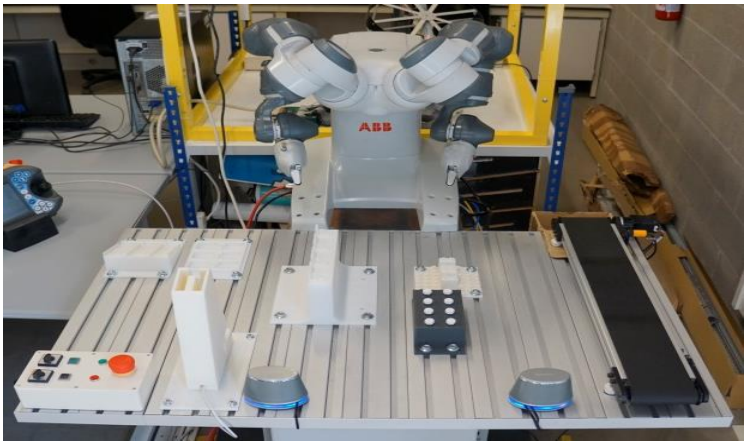


Ilustración 15. Escenario de partida

Tras analizar la disposición de los elementos que forman la célula de montaje y los movimientos que realiza el robot a lo largo del ensamblaje, es inviable introducir una plataforma que no interfiera con los proyectos previos, por lo que se ha optado por diseñar una mesa que sea sencilla de introducir cuando se utilice esta aplicación de entretenimiento, y de extraer cuando se realice el ensamblaje de las otras aplicaciones. (Ilustración 16).

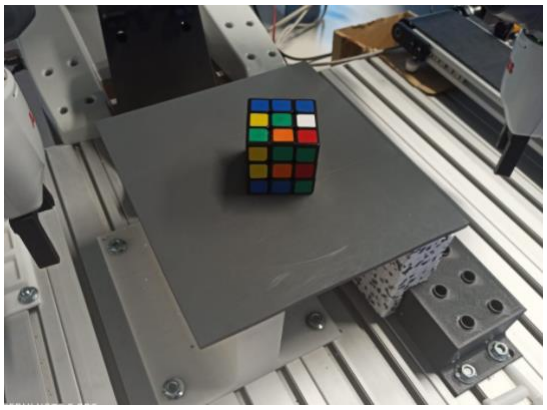


Ilustración 16. Mesa para el apoyo del cubo

Además de la mesa, se han modelado e impreso una serie de elementos auxiliares que permitan la consecución del proyecto.

5.2.2 Carcasa

La cámara está constituida únicamente por una placa base y una lente, por lo que ha sido necesario diseñar una carcasa que la proteja de golpes o polvo que puedan dañarla. Dado que el diseño de la carcasa se ha realizado desde cero, se ha modelado para que albergue también al objetivo y al domo de iluminación.

En la ilustración 17 se muestra el diseño CAD de la carcasa, la cual consta de dos partes: la base donde se apoyan la cámara, el objetivo y el domo (izquierda); y b) la tapa (derecha), diseñada de forma que ayude a sujetar los tres elementos sin permitir la movilidad de los mismos en el montaje final.

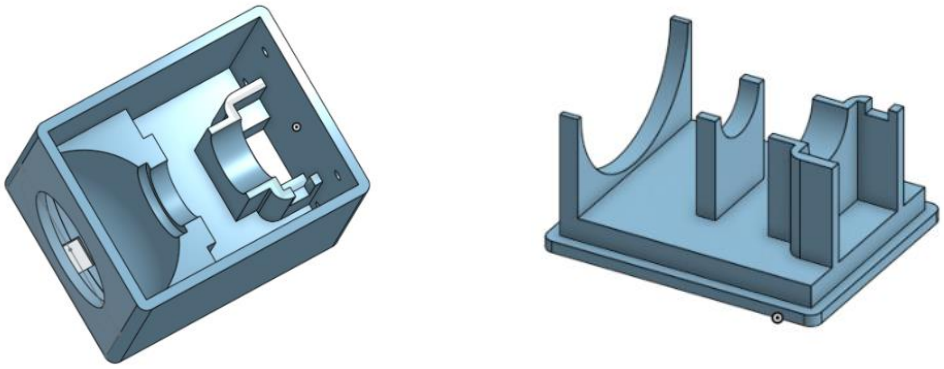


Ilustración 17. Modelado 3D de la carcasa y la tapa de la cámara

5.2.3 Fijación de la cámara en la célula

Tras diseñar la carcasa, se ha diseñado un amarre que sujete la misma sobre el robot en la posición correcta para tomar las fotografías. Debido al tamaño de la carcasa, se ha descartado la opción inicial de colocar la cámara entre los brazos del robot; y se ha modelado un amarre de mayor tamaño que permita colocar la carcasa sobre el robot, donde no interfiera con el movimiento de los brazos.

El amarre consta de dos partes, unidas entre sí mediante un acoplamiento prismático que permite regular la inclinación de la cámara. Además, se han realizado varios orificios a lo largo del amarre que permiten colocar la cámara a diferentes alturas, incluyendo una posición completamente horizontal que aporta una vista completa del espacio de trabajo. (Ilustración 18 y 19).

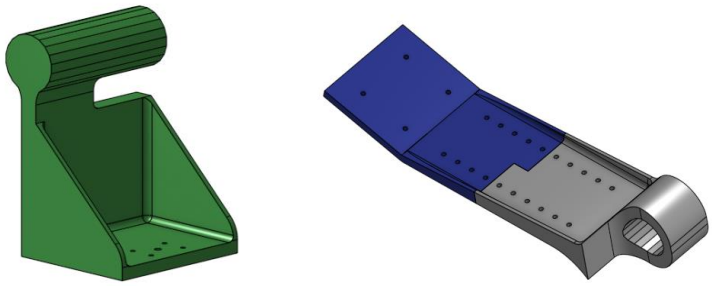


Ilustración 18. Modelado 3D del amarre de la cámara

Ilustración 19. Carcasa fijada al Yumi

5.2.4 Dedos de las pinzas

El robot Yumi incorpora dos pinzas colaborativas Smart Gripper diseñadas por ABB. Estas pinzas únicamente pueden abrirse 50 mm, lo cual imposibilita que puedan coger los cubos de rubik de 3x3 o 2x2, que miden 57 mm y 54 mm de lado, respectivamente. Por tanto, ha sido necesario diseñar unos nuevos dedos que permitan una mayor apertura y mejor adecuación de estos a cada parte móvil del cubo de rubik.

Dado que la parte del amarre entre la pinza y los dedos es la misma, se ha partido del diseño original de ABB y se ha modificado para lograr una apertura máxima de 65 mm que permita agarrar ambos cubos, 3x3 y 2x2, sin cambiar la pinza.

Por otro lado, para evitar colisiones al girar las caras del cubo, se ha modificado la geometría de los dedos, alargándolos al mismo tiempo, para permitir así una mayor distancia entre ambas pinzas al manipular el cubo. Finalmente, para darle una mayor robustez y evitar posibles rupturas en los puntos de unión que mayor esfuerzo tienen que soportar, se ha ensanchado tanto la base como la parte vertical, colocando radios de acuerdo en las zonas más críticas. En la ilustración 20 se muestra el plano CAD y la foto del de los dedos diseñados.

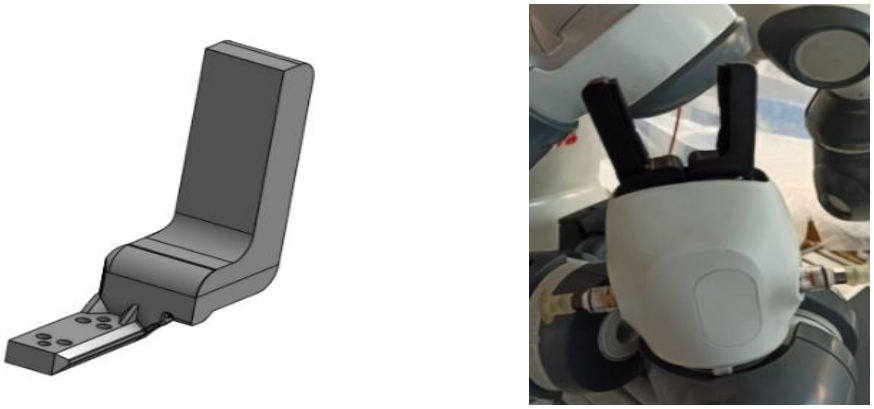


Ilustración 20. Modelado 3D del dedo de la pinza

5.3 Implementación del sistema

Tras la verificación del diseño, comprobando además que el robot dispone del espacio necesario para resolver el cubo sin interferir en la célula de montaje, se puede proceder a la implementación del sistema real.

El primer paso es la impresión y montaje de las piezas 3D. Para el amarre y la carcasa se ha utilizado una técnica de impresión por capas, mientras que, para los dedos de la pinza, se ha optado por una impresión con resina que aporta una mayor precisión.

Una vez impresos todos los elementos, se ha realizado el montaje, colocándose la cámara en la posición idónea para la captura de imágenes, así como la fuente de alimentación y demás elementos auxiliares que permitan su correcto funcionamiento. En la ilustración 21 se muestra una imagen definitiva del escenario.

Ilustración 21. Célula de montaje para la resolución del cubo de rubik

Con el montaje realizado puede comenzarse la implementación del sistema. El primer paso será la toma de imágenes que suministrarán de información al robot, por lo que se comienza con la programación de la cámara y el procesamiento de imágenes.

5.3.1 Configuración de la cámara

La cámara UI 5584LE-C-HQ (Infaimon) cuenta con una interfaz gigE (Gigabyte Ethernet) por lo que una vez alimentada, se conecta al ordenador con un cable de ethernet mediante su conector RJ45. Infaimon ofrece varias aplicaciones para poder acceder a la cámara por primera vez y configurarla de forma sencilla.

La primera de ellas es IDS Camera Manager, que permite gestionar todas las cámaras conectadas al equipo. La principal ventaja de esta aplicación es que encuentra la cámara automáticamente en el equipo y permite modificar su IP para conectarla a la red que se desee. Además de esto, ofrece información general sobre la cámara y su estado de funcionamiento.

En la ilustración 22 se muestra un pantallazo de la aplicación IDS Camera Manager donde puede apreciarse la lista de cámaras conectadas al equipo y parte de la información que aporta sobre ellas, así como las opciones que proporciona para configurarlas.

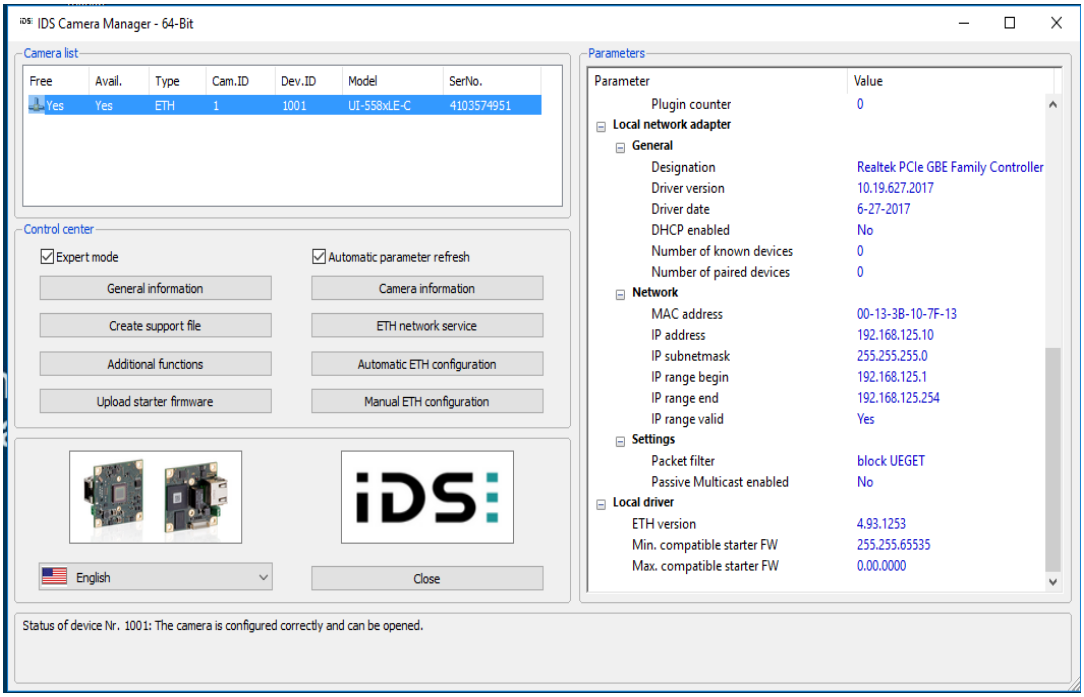


Ilustración 22. Interfaz IDS Camera Manager

Una vez la configuración de ethernet de la cámara esté lista, la aplicación Ueye cockpit (de Infaimon) será capaz de detectarla, y nos dará la opción de configurarla en función de las necesidades del proyecto. Cuando la cámara se enciende por primera vez, lo hace con la configuración estándar, mediante esta aplicación se pueden modificar los parámetros de resolución, modo de disparo, etc. y guardarlos como predeterminados.

En la ilustración 23 se muestra en interfaz de la aplicación Ueye Cockpit en la cual aparece en tiempo real lo que capta la cámara y en la parte superior y derecha todas las posibilidades que aporta para configurarla.

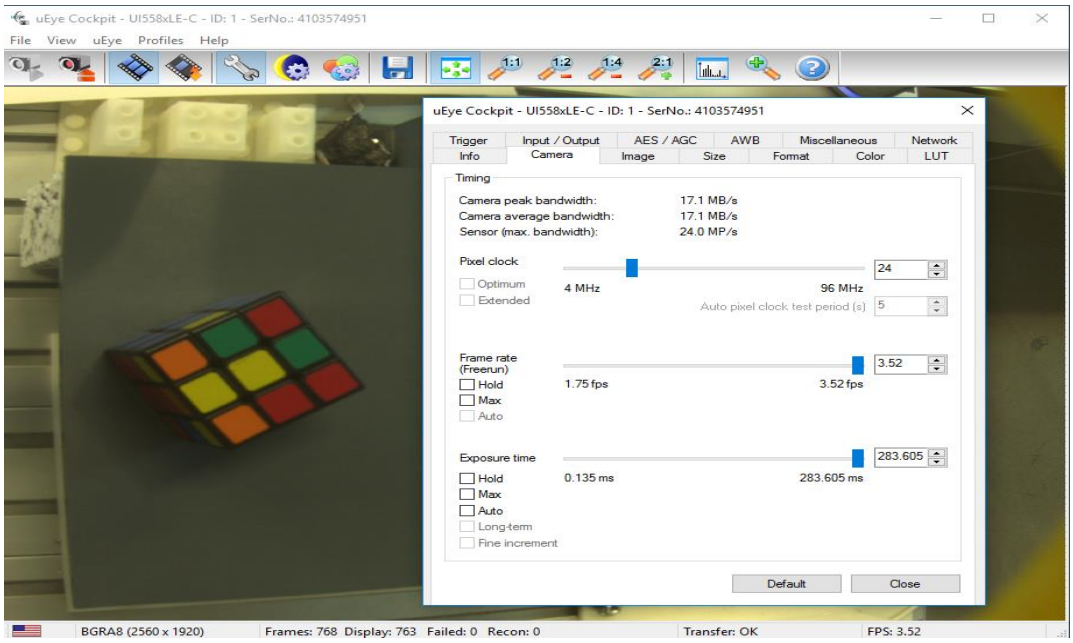


Ilustración 23. Interfaz Ueye Cockpit

5.3.2 Toma de imágenes

Con la cámara correctamente configurada, se procede a la captura/adquisión de imágenes. Como se ha explicado anteriormente, para este cometido se utiliza la herramienta Matlab, concretamente la *Image Adquisition Toolbox*.

Mediante esta *toolbox,* se puede acceder a todas las cámaras conectadas al equipo, y, al igual que en la aplicación Ueye Cockpit, configurar sus propiedades. Sin embargo, para hacerlo es necesario generar líneas de código que se ejecuten cada vez que la cámara se encienda, por ello se ha decidido hacerlo mediante Ueye Cockpit, pues una vez fijados los parámetros como predeterminados, la cámara siempre los adoptará directamente al encenderse.

La gran ventaja que aporta Matlab es que permite generar modos de disparo que activen la cámara, además de dar la posibilidad de elegir el número de imágenes por disparo y su formato o región de interés de cada fotografía, lo cual facilita su posterior procesamiento.

En la ilustración 24 se muestra el interfaz del Image Adquisition Toolbox. Como puede observarse, en la parte izquierda aparecen todas las cámaras conectadas al equipo y su información general. En la parte central se muestra lo que capta la cámara en cada momento y en la parte inferior los parámetros que permite configurar. Además, en la parte inferior derecha genera automáticamente el código que modifique dichos parámetros.

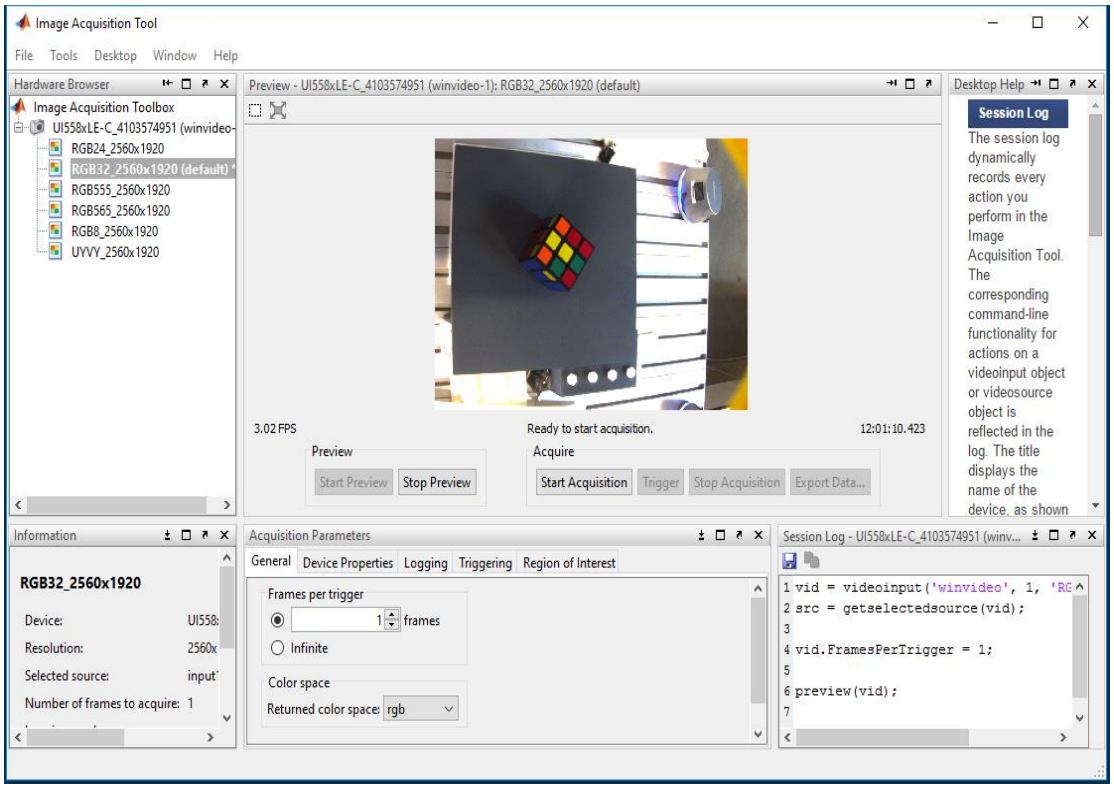


Ilustración 24. Interfaz Image Adquisition Toolbox

El sistema se ha configurado para que, a lo largo de un ciclo de ejecución se tomen siete fotografías. De esta forma se suministrará al robot la información completa necesaria, conociendo así el tipo de cubo (2x2 o 3x3) y la combinación de los colores que constituyen cada cara. Por tanto, la primera imagen muestra una vista superior de la zona de trabajo, de la cual se extrae el tipo de cubo que se va a resolver, su posición y su orientación. Las seis restantes corresponden a cada una de las caras del cubo, de las que se obtiene la forma en la que está mezclado

Al ejecutar el programa, el robot establece una comunicación TCP/IP con Matlab. Esta comunicación se detalla en el apartado 5.3.5 Conexión Matlab-Robot y sirve como señal de activación, para que la cámara capture la primera imagen.

Para ello, en primer lugar, se ha de inicializar la cámara mediante el comando *videoinput*, definiendo el canal y la resolución que se utilizará para la toma de imágenes. A continuación, se define el número de imágenes que se toman con cada disparo, y una vez hecho esto, se inicia la grabación mediante el comando *start(vid).* A partir de este momento, cámara comienza a grabar, por lo que, para tomar una única imagen, hay que ejecutar el comando *getsnapshot(vid)* que guardará la imagen tomada como ImagenCubo. Este comando toma la imagen rotada 90 grados y con un efecto espejo aplicado, por lo que se ejecutan los comandos *imrotate* y *fliplr* para solucionarlo. Por último, se detiene la grabación con el comando stop(vid) y se envía una señal al robot para hacerle saber que la imagen ha sido tomada.

En la ilustración 25 se muestra el código en Matlab.

```
tc=tcpip("192.168.125.1",14044); %Definición de la IP y el puerto para la coneción TCP/IP

fopen(tc); %Inicio de la conexión

num_foto=fread(tc); %Lectura de la variable enviada por el robot para la captura de la primera imaagen

%Toma_Imagenes;
vid = videoinput('winvideo', 1, 'RGB24_2560x1920'); %Inicialización de la cámara
vid.TriggerRepeat = 1 %Se fija el número de imágenes por trigger a 1

start(vid); % Comienzo de la captura de video

ImagenCubo = getsnapshot(vid);  %Toma de la imagen
X=ImagenCubo;
Y=imrotate(X,90); %Rotación 90 grados de la imagen
Z=fliplr(Y); %Aplicación de un filtro espejo a la imagen
imshow(Z); %Muestra de la imágen
stop(vid); %Finalización de la captura

fwrite(tc,foto); %Envío de señal al robot para informar de que la captura ha finalizado
```

Ilustración 25. Código Matlab para la toma de la primera imagen

La imagen obtenida se puede apreciar en la ilustración 26.

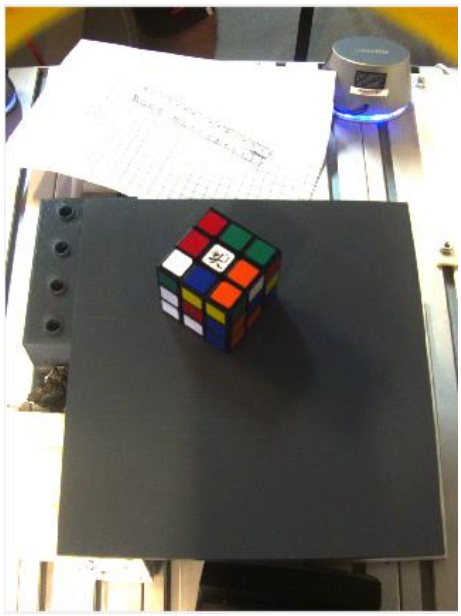


Ilustración 26. Imagen cubo en la mesa

Tras obtener la imagen se puede realizar su procesamiento del que se obtendrán el tipo de cubo con el que se trabaja, su posición en la mesa y su orientación. El método usado para el procesamiento se detalla en apartado siguiente, junto con el utilizado en las demás fotografías.

Una vez obtenida la posición del cubo, mediante la comunicación TCP/IP mencionada anteriormente, se envían las coordenadas a las que debe moverse uno de los brazos del robot para tomar el cubo (*punto de agarre del cubo*).

A partir de aquí se realiza una sincronización de acciones entre el robot y Matlab. De manera que, el robot coloca el cubo en la posición para la toma de la primera imagen; una vez ahí, envía la señal a Matlab que toma la fotografía y envía la señal de vuelta para que el robot coloque el

cubo en la posición para la siguiente imagen. Este proceso se repite seis veces, hasta que se obtienen las fotografías de cada una de las caras del cubo.

El código en Matlab es muy similar al de la toma de la primera imagen, con la diferencia de que, tras cada fotografía tomada, la variable Foto se incrementa en una unidad, de forma que, al enviarla al robot, este verifique que el número de la fotografía tomada es la correcta y pase a la siguiente posición. En la ilustración 27 puede observarse el código en Matlab.

```
num_foto=fread(tc); %Lectura de la señal que implica toma de una nueva imagen

start(vid); % Comienzo de la captura de video
CaraA = getsnapshot(vid);  %Toma de la imagen
X=CaraA;
Y=imrotate(X,90); %Rotación 90 grados de la imagen
Z=fliplr(Y); %Aplicación de un filtro espejo a la imagen
imshow(Z); %Muestra de la imágen
stop(vid); %Finalización de la captura
fwrite(tc,foto); %Envío de señal al robot para informar de que la captura finalizó

foto=foto+1 %Incremento de la variable asignada al número de la imagen
num_foto=fread(tc); %Lectura de la señal que implica toma de una nueva imagen

start(vid); %Inicio dela captura
CaraB = getsnapshot(vid); %Toma de la imagen
X=CaraB;
Y=imrotate(X,90); %Rotación 90 grados de la imagen
Z=fliplr(Y); %Aplicación de un filtro espejo a la imagen
imshow(Z); %Muestra de la imágen
stop(vid); %Finalización de la captura

fwrite(tc,foto); %Envío de señal al robot para informar de que la captura finalizó

foto=foto+1 %Incremento de la variable asignada al número de la imagen
```

Ilustración 27. Código Matlab para toma de imágenes de las caras del cubo

Respecto al código del robot, en él se diferencian el código del brazo izquierdo y del brazo derecho. El brazo derecho es el encargado de coger el cubo de la mesa, por lo que, una vez que lo tiene es el que abre la comunicación con Matlab y le hace saber que ha llegado a la posición para la toma de la primera imagen. Para ello, una vez definida la comunicación, le envía el mensaje “A” que hace referencia a la primera fotografía. Hecho esto, espera a que Matlab tome la imagen y le envíe la señal para moverse a la segunda posición.

Este proceso se repite para las tres primeras fotografías, a continuación, cierra la comunicación con Matlab e informa al brazo izquierdo que ha terminado de tomar las imágenes, para después colocarse en la posición para cambiar el cubo de brazo.

En la ilustración 28 se muestra el código del brazo derecho.

```
    !Crear comunicacion
    SocketCreate server1; !Definición del socket
    SocketBind server1, "192.168.1.125",14044; !Definición de la IP y el puerto
    SocketListen server1; !Esperar respuesta del cliente (Matlbab)
    SocketAccept server1,client1; !Creación de la comunicacíon

Foto1; !Llamada a rutima para llevar al cubo a la posición de la primera foto
SocketSend client1,\str:="A"; !enviar mensaje al cliente para hacerle saber que se ha llegado a la posición
SocketReceive client1,\RawData:=data; !Recibir mensaje del cliente
UnpackRawBytes data,1,foto\Ascii:=2; !Procesamiento del mensaje
oki:=StrtoVal(foto,numfoto);
WaitUntil numfoto=2; !Si el mensaje recibido es "2", se continúa el programa para tomar la segunda imagen

Foto2; !Llamada a rutima para llevar al cubo a la posición de la segunda foto
SocketSend client1,\str:="B"; !enviar mensaje al cliente para hacerle saber que se ha llegado a la posición
SocketReceive client1,\RawData:=data; !Recibir mensaje del cliente
UnpackRawBytes data,1,foto\Ascii:=2; !Procesamiento del mensaje
oki:=StrtoVal(foto,numfoto);
WaitUntil numfoto=3; !Si el mensaje recibido es "3", se continúa el programa para tomar la tercera imagen

Foto3; !Llamada a rutima para llevar al cubo a la posición de la tercera foto
SocketSend client1,\str:="C"; !enviar mensaje al cliente para hacerle saber que se ha llegado a la posición
SocketReceive client1,\RawData:=data; !Recibir mensaje del cliente
UnpackRawBytes data,1,foto\Ascii:=2; !Procesamiento del mensaje
oki:=StrtoVal(foto,numfoto);
WaitUntil numfoto=4; !Si el mensaje recibido es "4", se continúa el programa para tomar la cuarta imagen

SocketSend client1,\str:="Z";

SocketClose server1; !Cerrar comunicacion con el cliente

WaitSyncTask sync_RL_1, ListaTareas_RL; !Espera hasta que el brazo izquierdo tome las tres fotos
```

Ilustración 28. Código brazo derecho para la toma de imágenes de las caras de cubo

Al recibir esta señal, el brazo izquierdo inicia la comunicación con Matlab y se desplaza a la posición de intercambio del cubo, donde lo recoge. Una vez ahí, espera la confirmación de Matlab y se desplaza a la posición para tomar la cuarta fotografía. A partir de aquí, el proceso es análogo al del brazo derecho. En la ilustración 29 se detalla el código de los movimientos del brazo izquierdo para obtener la información de tres de las caras del cubo.

```
WaitSyncTask sync_RL_1, ListaTareas_RL; ! señal de sincronización para iniciar el proceso de toma de fotografías
    !Crear comunicacion
    SocketCreate server1; !Definición del socket
    SocketBind server1, "192.168.1.125",14043; !Definición de la IP y el puerto
    SocketListen server1; !Esperar respuesta del cliente (Matlbab)
    SocketAccept server1,client1; !Creación de l comunicación

Foto1; !Llamada a rutima para llevar al cubo a la posición de la primera foto
SocketSend client1,\str:="D"; !enviar mensaje al cliente para hacerle saber que se ha llegado a la posición
SocketReceive client1,\RawData:=data; !Recibir mensaje del cliente
UnpackRawBytes data,1,foto\Ascii:=2; !Procesamiento del mensaje
oki:=StrtoVal(foto,numfoto);
WaitUntil numfoto=5; !Si el mensaje recibido es "5", se continúa el programa para tomar la quinta imagen

Foto2; !Llamada a rutima para llevar al cubo a la posición de la segunda foto
SocketSend client1,\str:="E"; !enviar mensaje al cliente para hacerle saber que se ha llegado a la posición
SocketReceive client1,\RawData:=data; !Recibir mensaje del cliente
UnpackRawBytes data,1,foto\Ascii:=2; !Procesamiento del mensaje
oki:=StrtoVal(foto,numfoto);
WaitUntil numfoto=6; !Si el mensaje recibido es "6", se continúa el programa para tomar la sexta imagen

Foto3; !Llamada a rutima para llevar al cubo a la posición de la tercera foto
SocketSend client1,\str:="F"; !enviar mensaje al cliente para hacerle saber que se ha llegado a la posición
SocketReceive client1,\RawData:=data; !Recibir mensaje del cliente
UnpackRawBytes data,1,foto\Ascii:=2; !Procesamiento del mensaje
oki:=StrtoVal(foto,numfoto);
WaitUntil numfoto=7; !Si el mensaje recibido es "7", se continúa el programa para finalizar la toma de imágenes

!Cerrar comunicacion
SocketClose server1;

WaitSyncTask sync_RL_2, ListaTareas_RL; !Sincronización con brazo derecho
```

Ilustración 29. Código brazo izquierdo para la toma de imágenes de las caras de cubo

Cuando las seis fotografías han sido tomadas ambos brazos se mueven a la posición de sincronización donde esperarán a que Matlab les envíe los movimientos para resolver el cubo.

En la ilustración 30 puede apreciarse un diagrama con las acciones descritas.

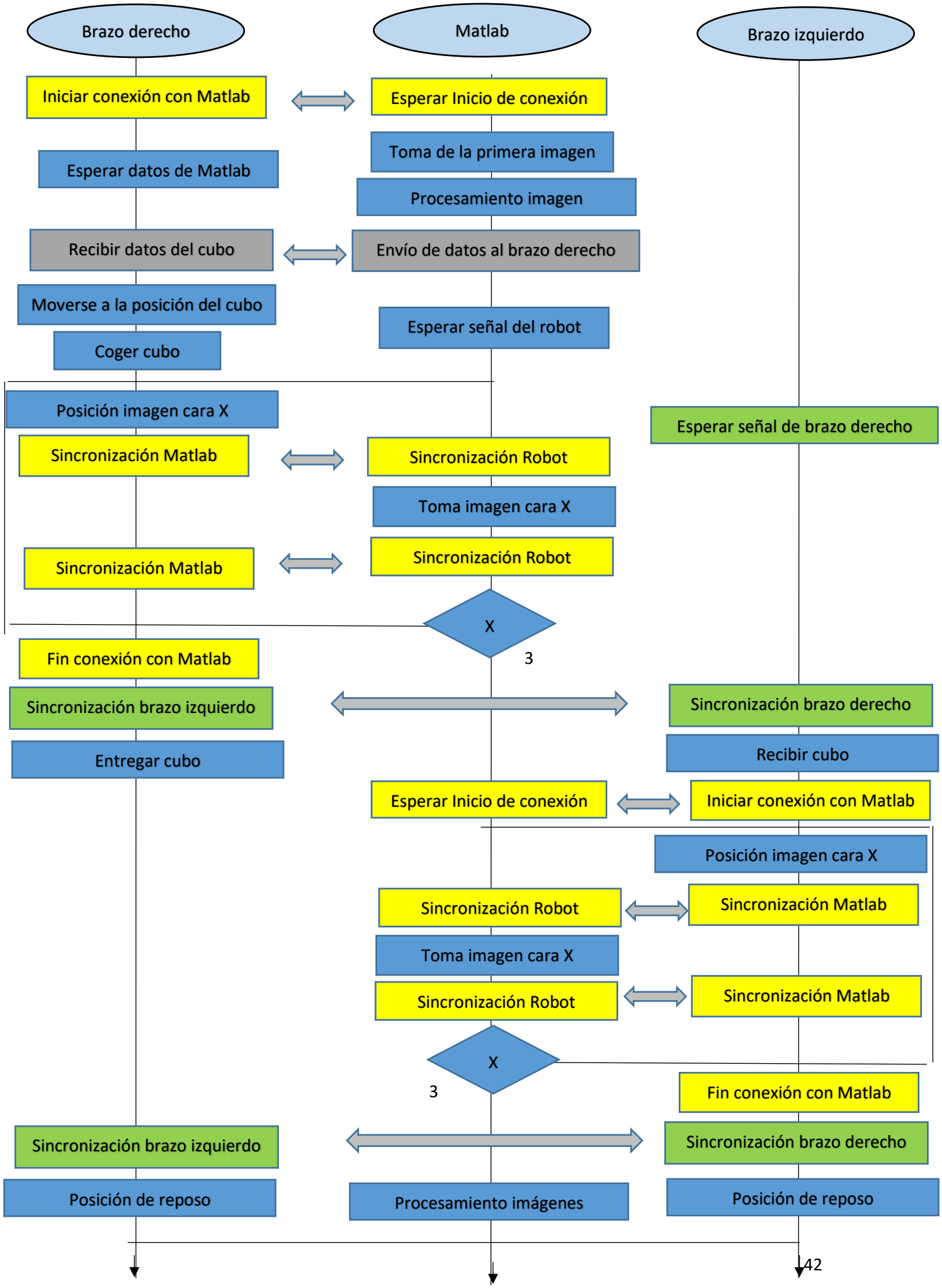


Ilustración 30. Diagrama toma de imágenes

5.3.3 Procesamiento de imágenes

Una vez tomadas las imágenes del cubo se lleva a cabo su procesamiento para extraer toda la información necesaria de ellas. Como se ha comentado en el apartado anterior, en primer lugar, se toma una imagen del cubo sobre la mesa y posteriormente, una de cada cara. La información a extraer de la primera imagen es diferente a la de las seis siguientes por lo que su procesamiento también lo es.

En primer lugar, se va a describir el procesamiento de cada una de las caras de cubo puesto que es más sencillo y permite explicar con mayor sencillez los conceptos y comandos utilizados.

5.3.3.1 Obtención del estado del cubo

Tras la toma de las imágenes del cubo, se realiza un procesamiento de los colores de cada cara que permita conocer la posición de cada una de las nueve piezas que las forman y, por ende, la forma en la que está mezclado el cubo.

Para el procesamiento del color se utiliza la función ColorThresholder, que genera una máscara que únicamente muestra los colores seleccionados, dejando el resto de la imagen en negro. Para la selección de los colores, permite trabajar con cuatro espacios de color: RGB, HSV, YcbCr y L*a*b*. Describe brevemente que es cada uno.

Se ha optado por trabajar en HSV (matiz, saturación, valor) ya que permite procesar los colores que forman el cubo de forma sencilla. Este espacio de color se representa mediante un cono invertido, en el cual, combinando los valores de matiz, saturación y valor, puede lograrse cualquier color. (Ilustración 31).

El matiz (en inglés Hue) se representa como un ángulo cuyo valor posible va desde 0 hasta 360 grados. Cada grado hace referencia a un color. La herramienta permite seleccionar un único grado o un rango, por lo que hace sencillo elegir los colores con los que trabajar.

La saturación (en inglés Saturation) representa la distancia al eje de brillo blanco-negro. Sus valores van de 0 a 1, y cuanto menores sean, mayor decolorado estará el color, adquiriendo una tonalidad grisácea.

Por último, el valor (en inglés Value) hace referencia a la altura en el eje blanco-negro. De nuevo, sus valores varían entre 0 y 1, siendo 0 negro y 1 blanco o cualquier otro color, dependiendo de la combinación entre matiz y saturación.

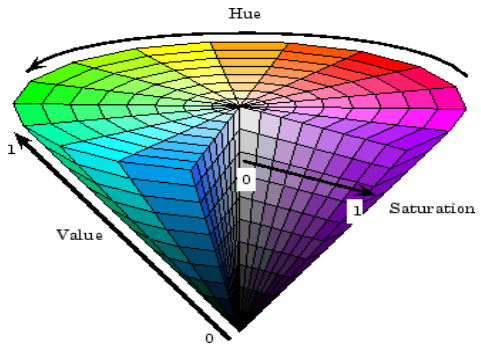


Ilustración 31. Cono del espacio de color HSV

Para seleccionar con precisión el color con el que se desea trabajar, la herramienta permite tanto variar las características del HSV como seleccionar el color directamente de la imagen, marcando la zona en la que se encuentra el color con el que se quiere trabajar. Una vez seleccionados los valores deseados se genera la máscara, que se puede exportar como función, de forma que puede usarse con otras imágenes.

En la ilustración 32, a la izquierda, se muestra una fotografía de la cara superior del cubo y a la derecha, la misma tras aplicarse la máscara. Como se aprecia, únicamente aparecen los objetos del color seleccionado en la máscara, que en este caso eran los verdes.

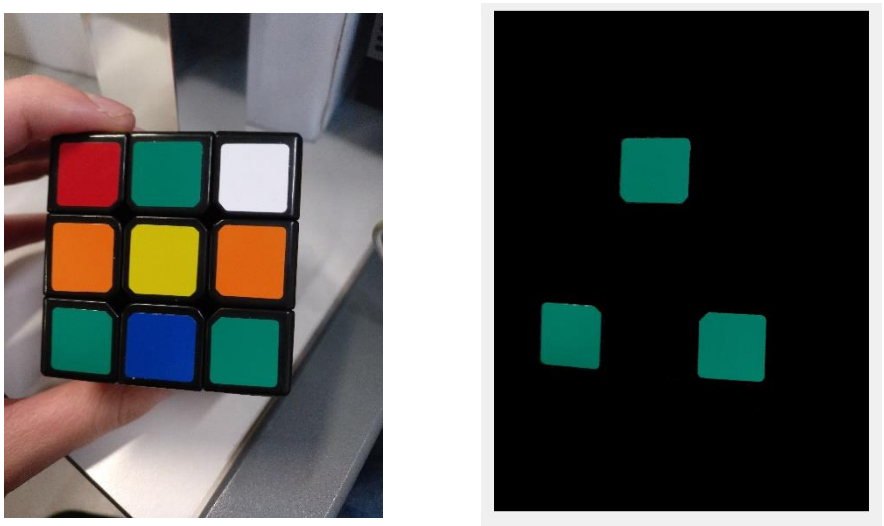


Ilustración 32. Cara del cubo antes y después de la aplicación de la máscara

A continuación, se debe de contar dichos objetos y conocer su posición en la cara. Para ello se utilizan las funciones *bwlabel* y *funcionprops*. Ambas funciones trabajan con imágenes binarias, por lo que es necesario binarizar la imagen obtenida. Afortunadamente, la propia herramienta permite hacerlo de ser necesario, obteniéndose el resultado que se muestra en la ilustración 27.

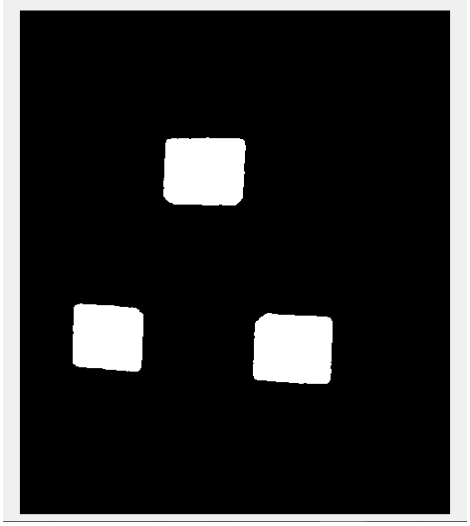


Ilustración 33. Cara del cubo con máscara y binarizada

Partiendo de la imagen binaria, la función *Bwlabel* la devuelve en forma de matriz, siendo todos los elementos de la misma ceros exceptuando los correspondientes a los objetos en blanco. Los elementos correspondientes al primer objeto serán unos, al segundo, doses y así sucesivamente. Además, también devuelve el número de objetos encontrados.

El comando *regionprops* parte de la matriz generada por bwlabel y devuelve el centroide y área de cada uno de los objetos encontrados. (Ilustración 34).

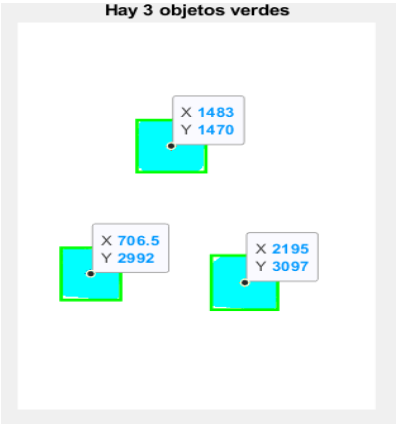


Ilustración 34. Localización de los centroides de las piezas

Esta subrutina se realiza con los seis colores que forman el cubo y para cada una de las caras. Los datos obtenidos de cada cara se almacenan en arrays dobles para cada color, donde se guardan las coordenadas del centroide de las piezas detectadas.

Para el ejemplo anterior, el array sería el que se muestra en la ilustración 35.

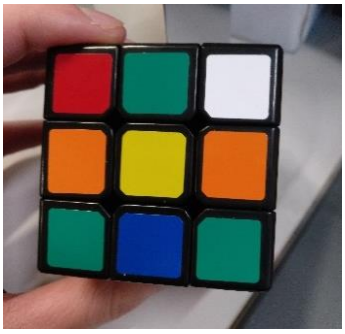
```
posItemGreen =

   1.0e+03 *

    0.7065    2.9917
    1.4830    1.4698
    2.1952    3.0969
```

Ilustración 35. Array de los centroides de las piezas

Una vez procesados todos los colores de la imagen, se genera la matriz Cara donde cada color se representa mediante un número. Siendo,

Rojo = 1; Azul = 2, Naranja = 3; Verde = 4; Blanco = 5 y Amarillo = 6.

Por tanto, la fotografía original de la cara se reduce a una matriz de dimensión 3x3, como se muestra en la ilustración 36.



```
Cara =

     1     4     5
     3     6     3
     4     2     4
```

Ilustración 36. Estado de una cara del cubo transformado en matriz

Por último, la información de cada Cara se pasa a un array multidimensional R que contiene toda la información del cubo mezclado. Dicho array está compuesto por seis matrices de 3x3, una por cara (Ilustración 37). Este array se enviará al algoritmo para la calcular la resolución del cubo.

```
R(:,:,1) =

     1     4     5
     4     1     1
     3     2     6

R(:,:,2) =

     6     1     3
     3     2     6
     6     3     4

R(:,:,3) =

     1     5     5
     2     3     2
     4     6     1

R(:,:,4) =

     3     1     2
     4     4     6
     3     1     2

R(:,:,5) =

     5     5     2
     2     5     5
     4     5     4

R(:,:,6) =

     6     6     5
     3     6     3
     1     4     2
```

Ilustración 37. Estado de cada cara del cubo transformado en matrices

5.3.3.2 Obtención del tipo, posición y orientación del cubo

Para la obtención de la posición y orientación del cubo de rubik en la mesa, se utiliza un método similar, sin embargo, se añade la dificultad de la perspectiva, ya que la mesa está inclinada con respecto a la cámara.

El objetivo es calcular la posición de las esquinas de la cara inferior del cubo, de forma que se pueda obtener la posición del su centro. (Ilustración 38).

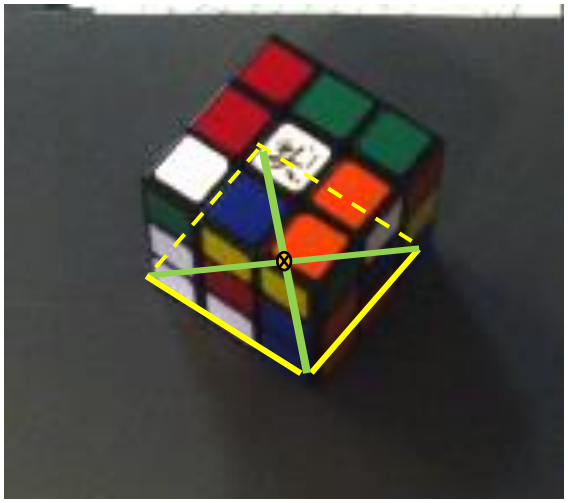


Ilustración 38. Cálculo del centro del cubo

Partiendo de la imagen obtenida por la cámara, se aplica una máscara de color que únicamente deje ver los colores que forman las caras del cubo, es decir, rojo, azul, naranja, verde, blanco y amarillo. Posteriormente, y al igual que en el apartado anterior, se binariza la imagen.

El resultado obtenido se puede observar en la ilustración 39:

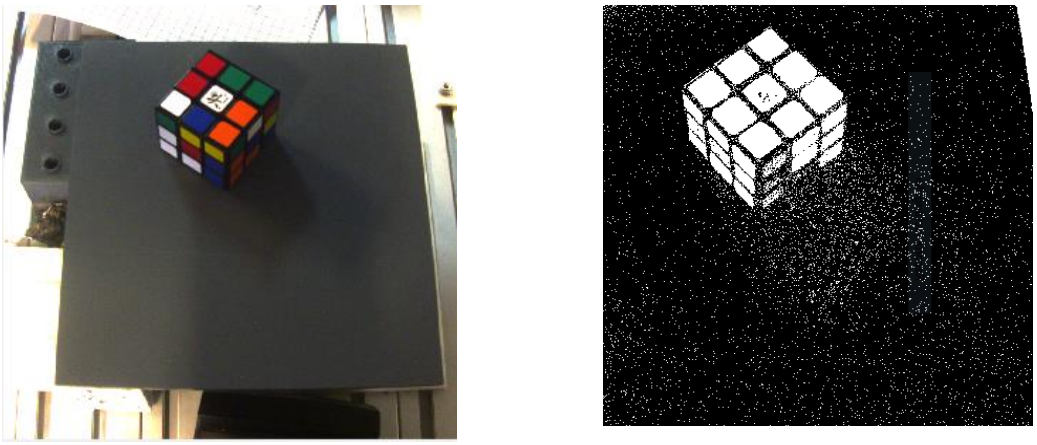


Ilustración 39. Binarización de la imagen del cubo en la mesa

Debido a la cantidad de colores que se incluyen en la máscara y a que el color de la mesa es gris, aparece gran cantidad de ruido en la imagen binaria, especialmente en las zonas más sombreadas. Para solucionarlo se aplica un filtro mediante el comando *strel,* y posteriormente, se ejecutan los comandos *cornelmetric* e *imregionalmax* que nos indicarán el contorno de cada uno de los objetos que forman la imagen binarizada.

Finalmente, se utilizará el comando *regionprops* que nos localizará cada uno de los puntos originados mediante los comandos anteriores y nos aportará su posición.

De esta forma podremos obtener las coordenadas de las esquinas del cubo que reposan sobre la mesa, y, por tanto, la posición de su centro su orientación. (Ilustración 40).

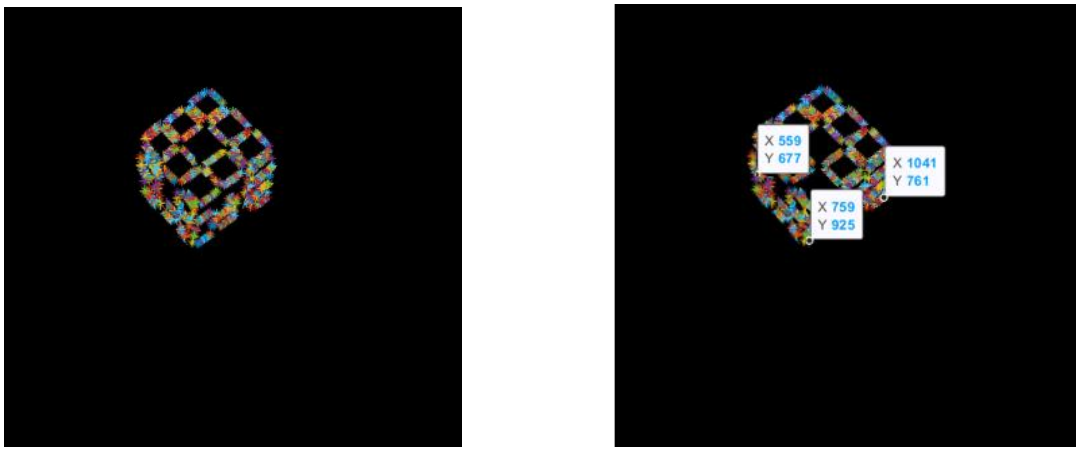


Ilustración 40. Cálculo de las esquinas del cubo en la mesa

A partir de aquí, mediante una plantilla, se han tomado referencias de la mesa y del sistema de referencia del robot. De esta forma, a partir de las coordenadas de las esquinas en la foto, se calculan las coordenadas a las que tiene que desplazarse el brazo derecho del robot para coger el cubo, y la orientación de la pinza para agarrarlo.

5.3.4 Algoritmo de resolución del cubo de rubik

Una vez obtenida la información sobre el estado del cubo mezclado, se procede a su resolución. En este apartado se aborda el algoritmo desde dos puntos de vista diferentes. Por un lado, planteando la resolución como lo haría un ser humano (método avanzado de Friedrich); y, por otro lado, aprovechando la capacidad de un ordenador y de un entorno como Matlab, calcular una solución de movimientos óptima (algoritmo de Joren Heit).

Ambos algoritmos parten del mismo punto, el array multidimensional R cuya obtención se detalla en el apartado anterior.

5.3.4.1 Algoritmo de Joren Heit

Joren Heit creó en 2011 un programa capaz de generar, simular y resolver cubos de rubik en Matlab. Para la generación del cubo, el programa permite tanto crear uno aleatoriamente, como introducir los colores de cada pieza a mano. Además, contiene una función para crear el cubo a partir de fotos tomadas por una webcam. Una vez generado el cubo, se pueden girar cada una de las caras manualmente para tratar de resolverlo, o bien utilizar alguno de los múltiples algoritmos de resolución que el programa aporta.

En la ilustración 41 se muestra la interfaz de su aplicación.

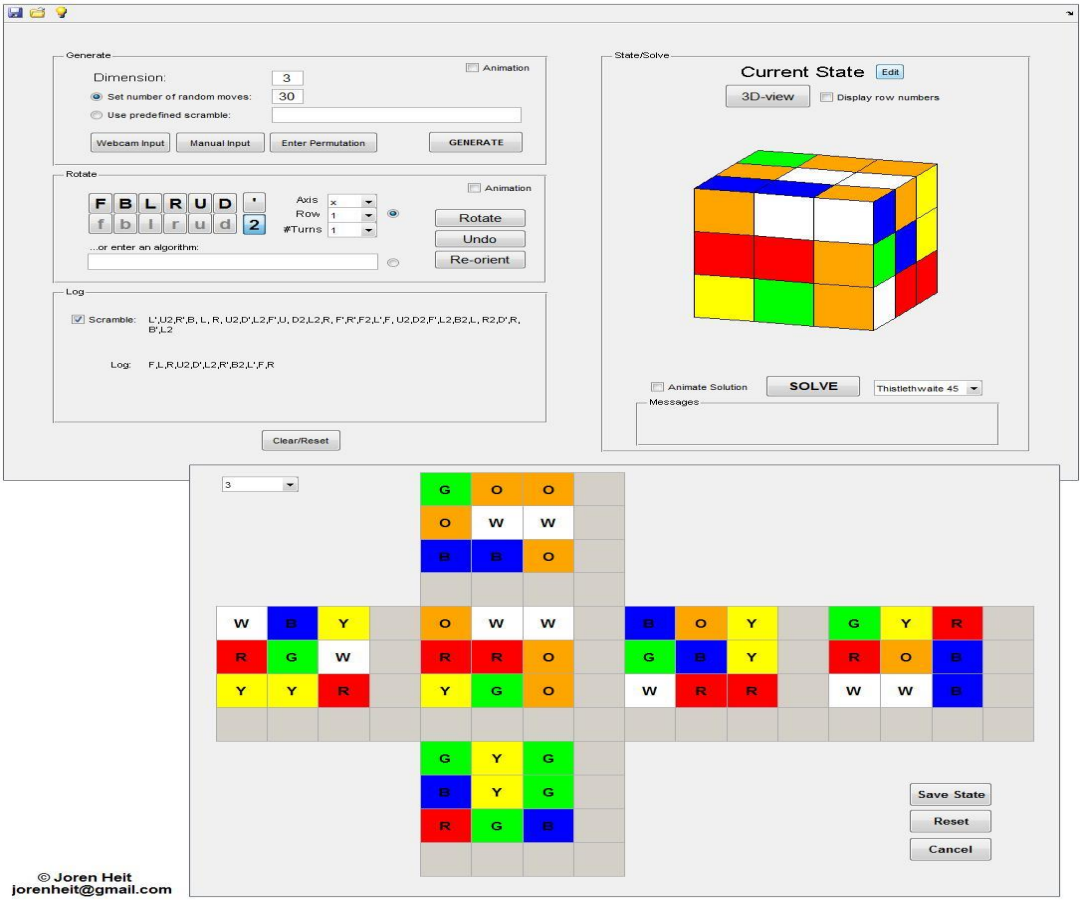


Ilustración 41. Aplicación de Joren Heit para la resolución del cubo de rubik en Matlab

Para esta aplicación, se ha partido del algoritmo de resolución Solve45. Este algoritmo asegura una resolución del cubo en menos de 45 movimientos, siendo la media de 31.

El algoritmo de Joren Heit está diseñado para ser utilizado por un ordenador. Se basa en unas tablas de poda generadas automáticamente y formadas por más de un millón de entradas que le permiten elegir la mejor solución en función de la forma en la que esté mezclado el cubo.

El algoritmo está basado en cuatro pasos:

(1) Orientación de los bordes
(2) Orientación de las esquinas y colocar los bordes de las caras derecha e izquierda
(3) Colocar los demás bordes y preparar las esquinas para su resolución
(4) Resolver el cubo

Para cada una de dichas fases el algoritmo calcula mediante las tablas de poda el número mínimo de movimientos que puede costarle pasar a la siguiente fase. Este número de movimientos mínimo se denomina d, y tras elegir un giro, se comprueba como varía. Si d disminuye el movimiento es correcto, ya que se encuentra un movimiento más cerca de la siguiente fase. Cuando d sea igual a 0, el algoritmo pasa de fase.

Así pues, partiendo del código de Joren se ha generado un algoritmo capaz de resolver el cubo con la información proveniente del procesamiento de imágenes, es decir, con el array R. Este algoritmo devuelve un array de celdas en las que cada una hacer referencia a un movimiento a realizar para resolver el cubo.

En la ilustración 42 se muestra el array solución:

```
sol =

  1×28 cell array

  Columns 1 through 12

    {'L'}    {'R2'}    {'F'}    {'U'}    {'L2'}    {'D'}    {'L''}    {'D2'}    {'B''}    {'L''}    {'R2'}    {'F'}

  Columns 13 through 23

    {'L'}    {'U2'}    {'L'}    {'D2'}    {'L''}    {'U2'}    {'L''}    {'F2'}    {'L2'}    {'U2'}    {'F2'}

  Columns 24 through 28

    {'U2'}    {'L2'}    {'U2'}    {'L2'}    {'B2'}
```

Ilustración 42. Solución del método de Joren Heit

Este array sigue la notación tradicional utilizada en los algoritmos del cubo de rubik. Cada letra hace referencia a una cara del cubo, siendo F la frontal, R la derecha, B la trasera, L la izquierda, U la superior y D la inferior. Si la letra aparece sola significa que hay que realizar un giro horario; si aparece seguida de un apostrofe el giro ha de ser anti horario, y, por último, si viene seguida por un 2, el giro será doble.

La notación es intuitiva para las personas, no obstante, es compleja de procesar para el robot por lo que se ha modificado, sustituyendo cada celda por números enteros siguiendo método mostrado en la tabla 2.

	Giro horario	Giro anti horario	Giro doble
Cara F	10	11	12
Cara R	20	21	22
Cara B	30	31	32
Cara L	40	41	42
Cara U	50	51	52
Cara D	60	61	62

Tabla 2. Método para transformar los giros de las caras del cubo a números enteros

Como puede observarse, cada celda se ha sustituido por un número entero de dos cifras. El primer número hace referencia a la cara, mientras que el segundo al tipo de giro a realizar.

Para realizar el cambio se ha recorrido el array solución leyendo cada celda y generando el array movimientosR1 en función de su contenido. El resultado se muestra en la ilustración 43 y el código puede encontrarse en el anexo II.

```
movimientosR1 =

  Columns 1 through 19

    40    22    10    50    42    60    41    62    31    41    22    10    40    52    40    62    41    52    41

  Columns 20 through 28

    12    42    52    12    52    42    52    42    32
```

Ilustración 43. Solución del algoritmo en números enteros

Este array es el que se enviará al robot tal y como se describe en el siguiente apartado.

5.3.4.2 Algoritmo avanzado de Friedrich

El algoritmo de Joren Heit es prácticamente imposible de reproducir por un humano. Aparte de la necesidad de generar una tabla de poda, el hecho de calcular el siguiente movimiento óptimo tomaría demasiado tiempo. Por ello, se han desarrollado otro tipo de algoritmos que permitan a las personas resolver el cubo lo más rápido posible.

Uno de los métodos más extendidos es el método avanzado de Friedich. Este algoritmo es una mejora del método layer by layer (Capa por capa), que es el algoritmo que habitualmente utilizan los principiantes.

El algoritmo, consiste en cuatro pasos. (ilustración 44).

(1) Resolución de las aristas blancas, creando la cruz blanca.
(2) Método F2L (First Two Layers) para resolver las dos primeras caras a la vez.
(3) Orientación de la última cara mediante el método OLL (Orientation of Last Layer).
(4) Resolución de la última cara mediante los algoritmos PLL (Permutation of Last Layer).

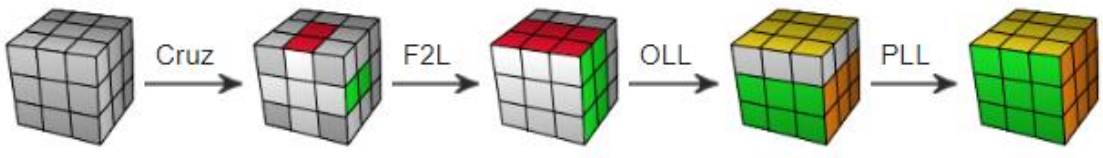


Ilustración 44. Pasos para resolver el cubo de rubik por el método de Friedrich avanzado

Debido a que el Yumi está diseñado para colaborar con humanos, se ha programado este algoritmo en Matlab, de forma que se podrá comparar su velocidad de resolución del cubo con la de una persona.

Para ello, se ha partido del array de matrices (R) generado por el procesamiento de imágenes y se han programado cada uno de los posibles giros y rotaciones que pueden realizarse para resolver el cubo.

Una vez hecho esto, se han programado cada una de las cuatro fases de la resolución del cubo, escogiendo para cada situación la solución más oportuna según el método de Friedrich avanzado. Además, se ha generado una rutina que muestre el estado del cubo tanto en 3D como desplegado al acabar cada paso. En la ilustración 45 se puede apreciar el proceso de resolución del cubo de rubik de 3x3.

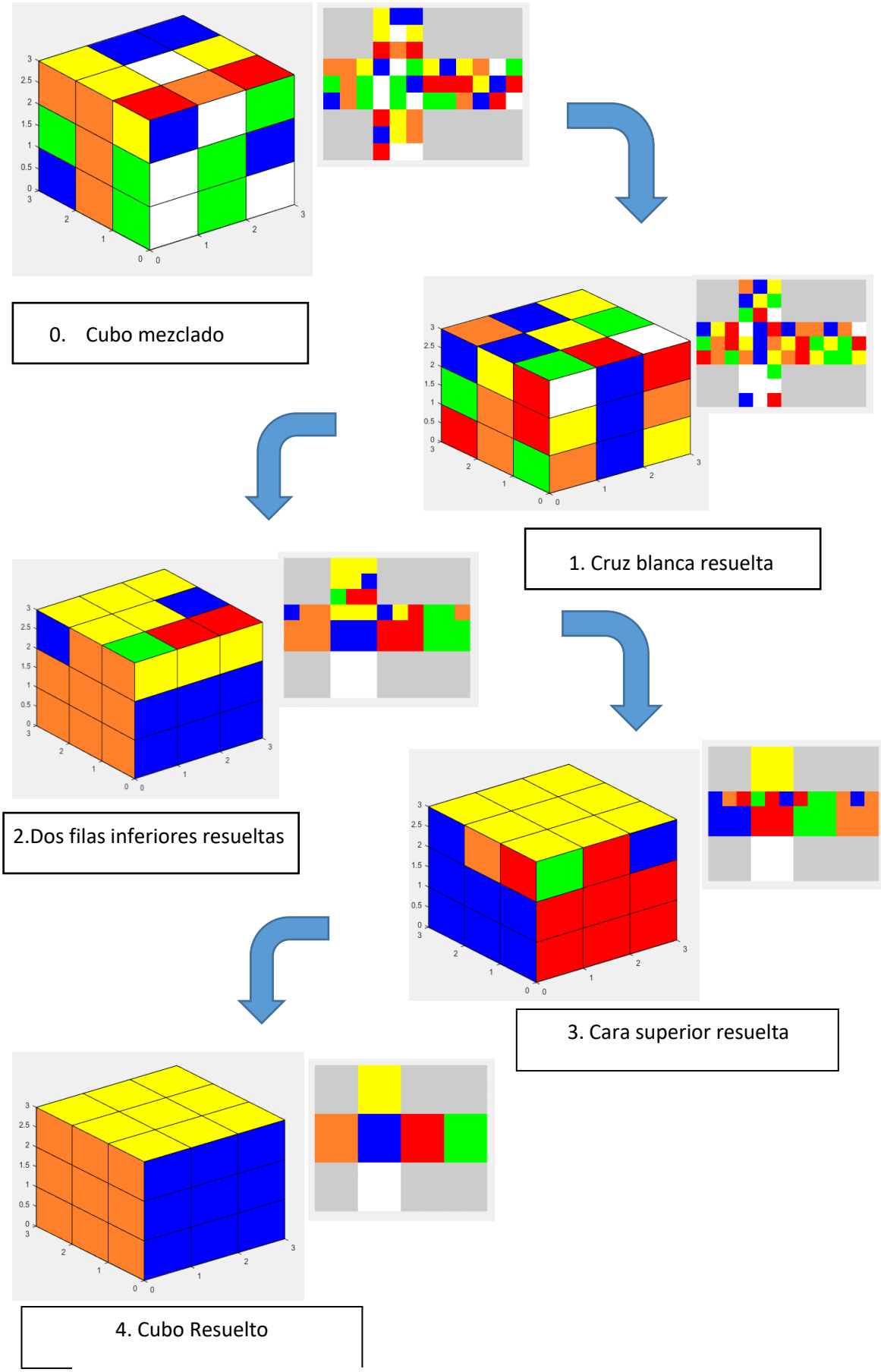


Ilustración 45. Esquema de la resolución del cubo de rubik en Matlab por el método de Friedrich avanzado

5.3.4.3 Cubo de rubik de 2x2

Con respecto al cubo de rubik de 2x2, se ha utilizado para su resolución un método análogo al del cubo de 3x3. Sin embargo, en este caso, el algoritmo es considerablemente más sencillo, ya que puede considerarse una simplificación del cubo de 3x3 en el que no existen bordes, siendo todas las piezas esquinas.

Esto implica que la tabla de poda generada es más sencilla y que únicamente es necesario un paso para resolverlo. Al estudiar la tabla de poda generada, se observa que el número máximo de movimientos necesarios para resolver el cubo es de once, independientemente de cómo esté mezclado. Siendo 8,8 la media de movimientos necesarios.

En la ilustración 46 se muestra la resolución del cubo en Matlab, mientras que el código completo se encuentra en el anexo II.

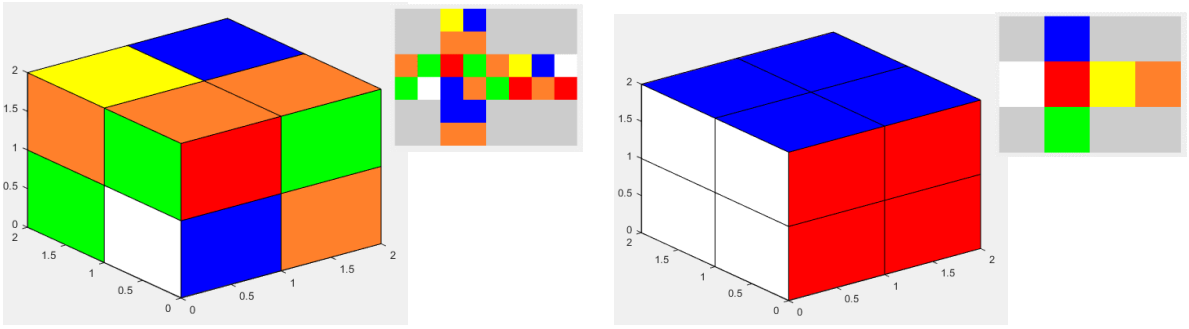


Ilustración 46. Resolución del cubo de 2x2 en Matlab

Al igual que en el caso anterior, este método está diseñado para ser utilizado por un ordenador, pero en las últimas décadas se han desarrollado diversos algoritmos para que cualquier persona pueda resolverlo. Sin embargo, al tratarse de una simplificación del cubo de 3x3, se ha optado por no programar ninguno de ellos en Matlab.

5.3.5 Conexión Matlab - Robot

Tras obtener los movimientos codificados necesarios para resolver el cubo, hay que enviárselos al robot YUMI para que los brazos ejecuten los movimientos correspondientes. En este apartado se desarrolla la comunicación entre Matlab y el robot que permita el intercambio de datos entre ambas partes.

Se ha optado por una comunicación mediante protocolos TCP/IP, que agrupan un conjunto de protocolos de internet que permiten la interconectividad entre diferentes arquitecturas hardware y software conectadas entre sí mediante redes. Toma su nombre de sus protocolos más característicos, el protocolo TCP (Transmission Control Protocol) que proporciona un transporte fiable de los datos, es decir, garantiza que se entregarán y que lo harán en el orden correcto; y del protocolo IP (Internet Protocol), que se encarga de enviar los datos a otros dispositivos conectados a la red.

Para lograr la comunicación es necesario definir los sockets de internet, que son las interfaces que permiten la comunicación y el intercambio de datos entre aplicaciones, tanto de distintos ordenadores, como dentro del mismo. Están formados por un puerto y una dirección IP.

La comunicación TCP/IP se basa en una relación cliente servidor. Se ha definido al robot como servidor y a Matlab como cliente. Por tanto, es el robot quien abre la comunicación y Matlab quien la acepta cuando ha terminado de procesar la primera de las imágenes.

A continuación, se describe el código utilizado en Matlab y en el robot para el establecimiento de la conexión. (ilustración 47).

Como se ha explicado, el robot es el que actúa como servidor y, por tanto, el que abre la comunicación. Antes de comenzar a programarla es necesario definir las variables del servidor y del cliente que serán del tipo socketdev y a los que hemos nombrado server1 y client1.

Hecho esto, el primer paso es la creación del socket vinculado a dicho servidor mediante el comando SocketCreate. Posteriormente ha de definirse el socket, para lo cual hay que especificar la IP y puerto que se utilizará para la comunicación. Una vez el socket está definido entabla comunicación con el cliente, y cuando este acepte, la comunicación estará correctamente establecida.

```
SocketCreate server1; !Creación del socket vinculado al servidor Server1
SocketBind server1, "192.168.125.1",14044; !Definición del puerto y la IP del socket
SocketListen server1; !Espera a la confirmación del cliente
SocketAccept server1,client1; !Creación de la comunicación
```

Ilustración 47. Código robot establecimiento conexión TCPIP

Para que Matlab pueda aceptar la conexión, simplemente es necesario definir el socket que se va a utilizar. La forma de hacerlo es mediante el comando tcpip seguido de la IP y puerto del socket.

Por último, al ejecutarse el comando fopen esperará y aceptará la comunicación con el servidor. (Ilustración 48).

```
tc=tcpip("192.168.125.1",14044); %Definición de la IP y el puerto para la coneción TCP/IP
fopen(tc); %Inicio de la conexión
```

Ilustración 48. Código Matlab establecimiento conexión TCPIP

5.3.6. Envío de movimientos

Una vez establecida la comunicación, se procede al envío de los datos al robot. En primer lugar, es necesario contar los movimientos a enviar y guardar la información en la variable num_mov. Esto es importante ya que el robot no es capaz de procesar mensajes de más de 80 caracteres. Hay que tener en cuenta que cada movimiento, excepto el primero y el último, ocupa tres caracteres (dos del número entero y uno del espacio de separación), por lo que en caso de que sean necesarios más de 26 movimientos para resolver el cubo, habrá que partir el array y realizar dos envíos.

En la ilustración 49 se muestra el código utilizado para dicha partición, en caso de ser necesaria.

```
if length(movimientosR1) > 26
    j=1;
    for i=27:length(movimientosR1)
        movimientosR2(j)=movimientosR1(i);
        j=j+1;
    end
    for i=length(movimientosR1):-1:27
        movimientosR1(i)=[];
    end
end
```

Ilustración 49. Partición de los arrays de movimientos

El siguiente paso es enviar el número de movimientos al robot de forma que sepa la cantidad de caracteres que tiene que recibir y procesar.

Para enviar un dato desde Matlab se utiliza el comando *fwrite* seguido del nombre de la comunicación y del dato a enviar. Cabe destacar que para enviar cualquier tipo de dato es necesario transformarlo al tipo *string* previamente.

Por su parte, el robot recibirá la información mediante el comando *SocketReceive* y *UnpackRawBytes* donde habrá que definir la conexión y el número de caracteres a recibir. Posteriormente, deshará el cambio de tipo de dato con el comando *StrToVal*

En la ilustración 50 se muestra el código utilizado para el envío de la variable num_mov.

```
datos=mat2str(num_mov);
fwrite(tc,datos);
```

```
SocketReceive client1,\RawData:=data;
UnpackRawBytes data,1,Recmov\Ascii:=2;
oki:=StrToVal(Recmov,nummov);
```

Ilustración 50. Código para envío de datos en Matlab y Rapid

Hecho esto se procede al envío de los movimientos. Como se ha comentado, en caso de ser menos de 26 movimientos se realiza un único envío, mientras que si son más se harán dos, dejando una breve pausa entre ellos. El código es análogo al utilizado para el envío del número de movimientos.

Una vez el robot dispone de la información, recorre los arrays, almacenando los movimientos en el array vec. (Ilustración 51).

```
FOR j FROM 1 TO nummov DO
 part:=StrPart(Recmov1,B,2);
 ok:=StrToVal(part,vec{j});
 B:=B+3;
ENDFOR
```

Ilustración 51. Procesamiento de los movimientos en rapid

5.3.7 Simulación de la estación en Robotstudio

Una vez que el robot dispone de los movimientos, únicamente queda generar las trayectorias que le permitan girar las caras del cubo y resolverlo. Para ello, se utilizará el programa RobotStudio.

RobotStudio es un software creado por ABB que permite la programación y simulación de los robots de la propia marca. Además, añade la posibilidad de importar o diseñar piezas que acompañen al robot, pudiéndose de esta forma simular células de montaje completas.

La creación de la estación en RobotStudio permite, además de programar el robot de una forma sencilla e intuitiva, comprobar que el robot es capaz de alcanzar todas las posiciones y de realizar los movimientos correctamente, corroborando así que el diseño de la célula es correcto antes de realizar la implantación en el sistema real.

5.3.7.1 Creación de la estación en RobotStudio

Para la generación de la célula, se parte de la célula de montaje de camiones de lego y de engranajes realizada durante el curso 2018/2019 (ilustración 52), que cohabitarán en el mismo espacio de trabajo que la que aquí se plantea de resolución de cubo de rubik.

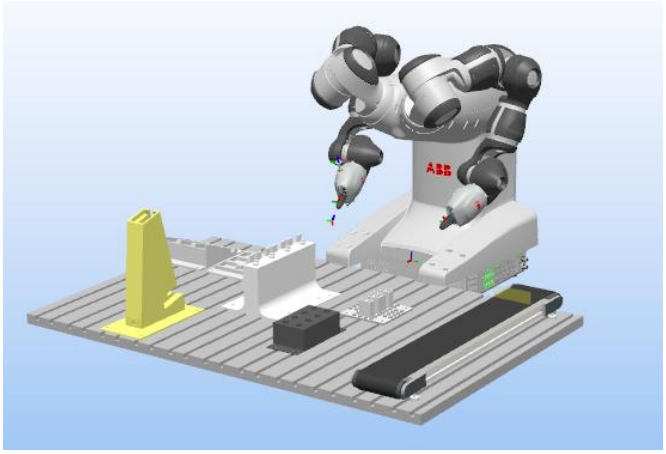


Ilustración 52. Célula de montaje del curso 2018/2019 en RobotStudio

Tras el análisis del espacio de trabajo realizado en el primer apartado y el diseño de las piezas en 3D, se importan y colocan los modelos CAD. La estación completa puede apreciarse en la ilustración 53.

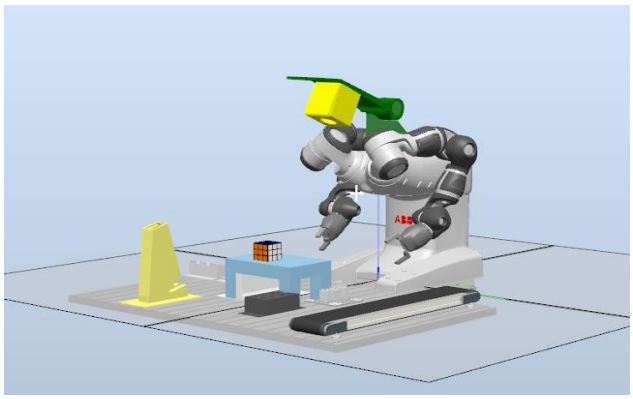


Ilustración 53. Célula de montaje para resolución del cubo de rubik en RobotStudio

5.3.7.2 Creación de componentes inteligentes

Los componentes inteligentes son mecanismos controlados mediante entradas y salidas digitales que dotan a las simulaciones de un mayor realismo. En este proyecto se utilizarán para controlar las pinzas del robot y el cubo, dándoles la capacidad de abrirse y cerrarse, así como de coger y mover objetos.

RobotStudio permite importar las pinzas Smart Gripper desarrolladas por ABB desde su biblioteca. Estas pinzas están programadas de antemano, por lo que únicamente hay que fijar las posiciones de apertura y cierre y las señales digitales que hagan que la pinza se abra o se cierre. Sin embargo, como se ha comentado anteriormente, las pinzas originales del Yumi no son lo suficientemente grandes ni adecuadas como para agarrar el cubo de rubik, por lo que hay que desarrollar el mecanismo de las pinzas desde cero.

Para ello, en lugar de importar las pinzas desde la biblioteca, se importan los CADs de cada uno de los componentes de la pinza, a excepción de los dedos, que se sustituirán por los creados en el apartado 5.2.4.

Una vez importados, hay que generar los mecanismos que permitirán la apertura y cierre de los dedos, para lo cual RobotStudio tiene un apartado específico.

El primer paso es darle un nombre a mecanismo y seleccionar su tipo de entre las opciones existentes. En este caso, al ser una pinza, el mecanismo será una herramienta. (Ilustración 54).

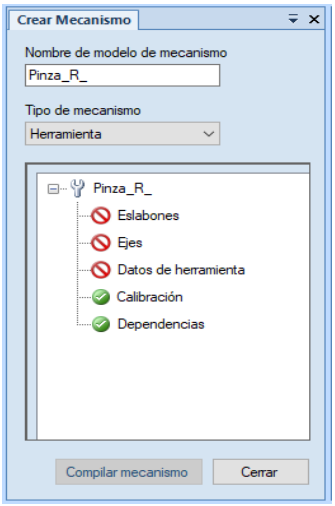


Ilustración 54. Apartados a definir para generar mecanismos en RobotStudio

Como se aprecia en la ilustración 54, se deben configurar tres subapartados que definen las características y movimiento del mecanismo. El primero de ellos son los eslabones o elementos que lo van a componer. Para definirlos es suficiente con seleccionar de entre todos los objetos de la estación, los componentes de cada eslabón, tal y como se muestra en la ilustración 55. En este caso, los eslabones serán los dedos y la propia pinza, que actuará como eslabón base.

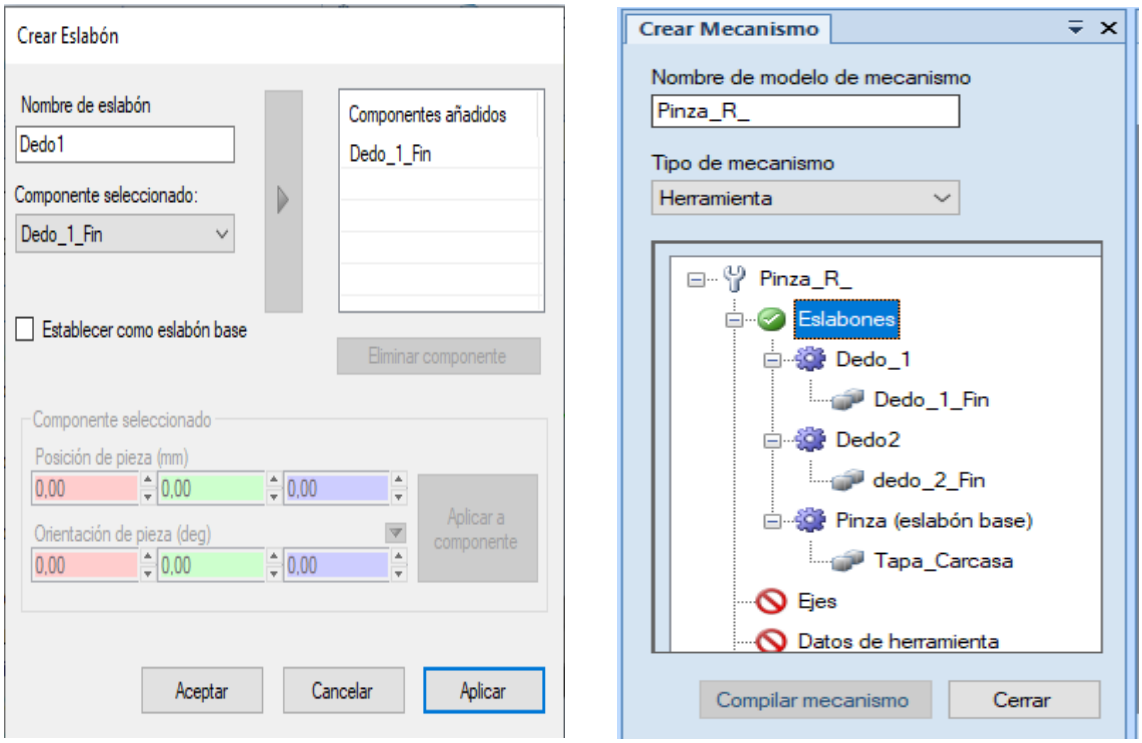


Ilustración 55. Selección de eslabones para la creación de mecanismos en RobotStudio

Una vez los componentes están seleccionados es necesario definir sus ejes de movimiento. Como se aprecia en la ilustración 56 se hace seleccionando dos puntos pertenecientes al eje y posteriormente introduciendo los límites del movimiento de cada eslabón.

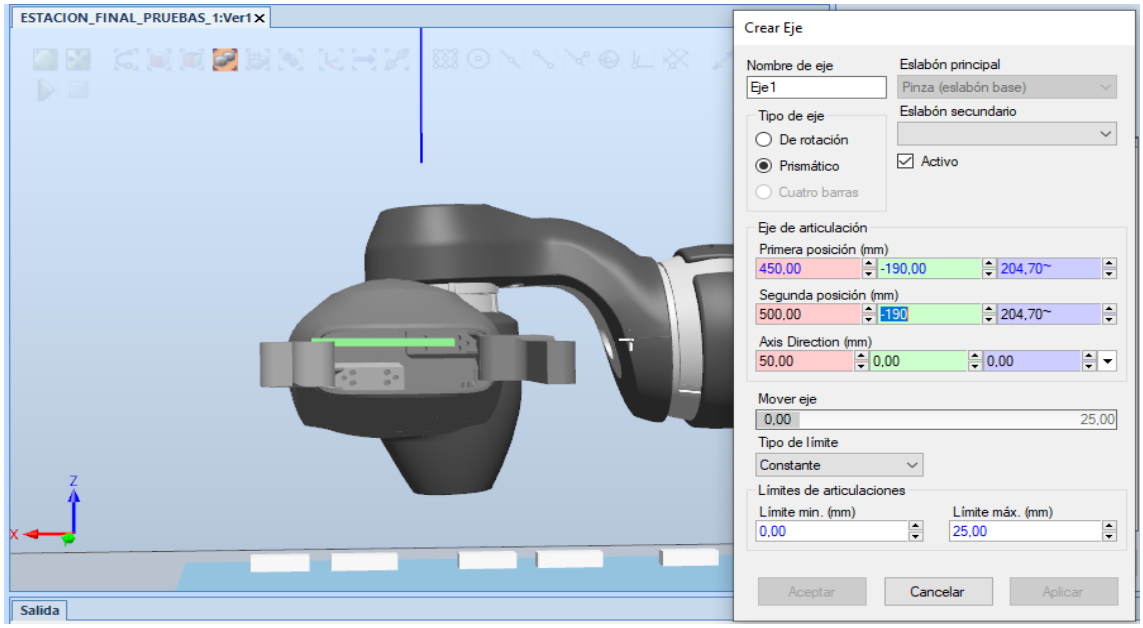


Ilustración 56. Definición de los ejes de los mecanismos en RobotStudio

Por último, es necesario especificar la masa, centro de gravedad y centro de inercia del mecanismo para que el robot lo tenga en cuenta a la hora de generar las trayectorias.

Con los mecanismos que forman las pinzas debidamente configurados se puede comenzar a la creación de los componentes inteligentes que generen los movimientos de las pinzas. Para ello se selecciona la opción modificar mecanismo que aparece al hacer clic derecho en cada pinza.

Al hacerlo aparece una pestaña como la que se aprecia en la ilustración 57 en la cual se puede definir cada una de las posiciones de la pinza, es decir, la posición de los eslabones cuando la pinza esté abierta, cerrada o en alguna posición intermedia que interese.

Para que la pinza se mueva a cualquiera de las poses definidas, bastará con vincularla a una salida digital y activarla.

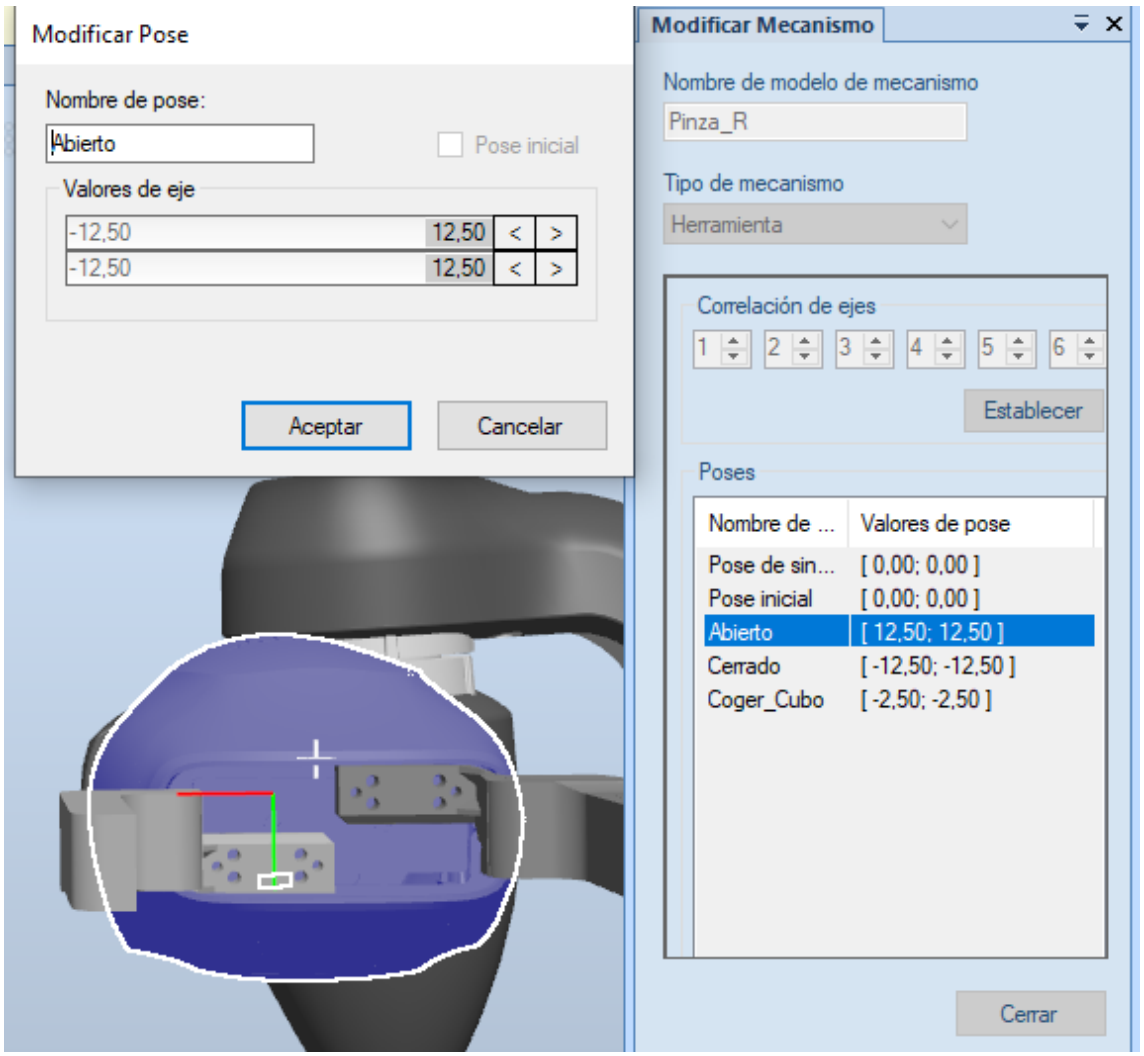


Ilustración 57. Definición de las posiciones de las pinzas en RobotStudio

5.3.7.3 Generación del cubo de rubik

Para realizar una simulación más completa, se ha generado un cubo de rubik funcional en RobotStudio. El cubo, al igual que uno real, está formado por 26 piezas de forma cúbica de 19 mm de lado dispuestos en forma de cubo de 3x3x3 y con un hueco en medio, donde se colocaría el mecanismo que permite el giro de cada una de las caras. En la ilustración 58, a la izquierda se muestra el cubo real desmontado y a la derecha el generado en RobotStudio.

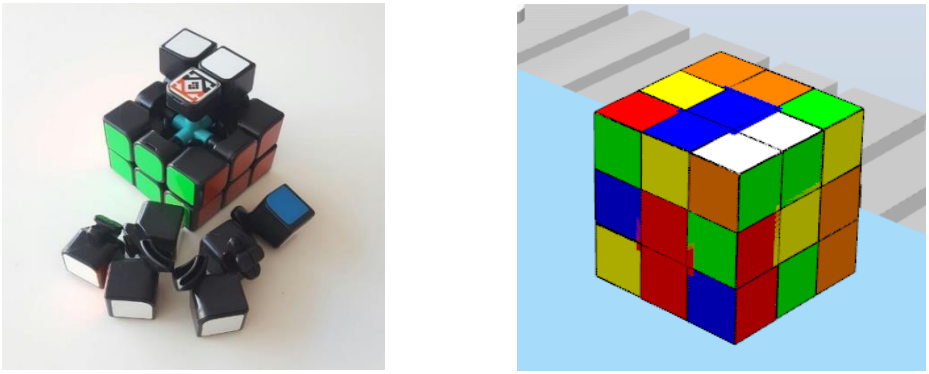


Ilustración 58. Mecanismo del cubo de rubik real frente al cubo en RobotStudio

Debido a la dificultad para realizar un mecanismo como el de un cubo de rubik real mediante RobotStudio, se ha optado por un método diferente basado en sensores de colisión. Las piezas centrales de cada cara tienen un sensor de colisión con respecto a cada una de las otras 25 piezas. Cada sensor está ligado a un *attacher* que, cuando se activa, las une. En la ilustración 59 se muestra la lógica de una de las caras del cubo, pudiéndose apreciar las conexiones entre los sensores de colisión y los *attachers.*

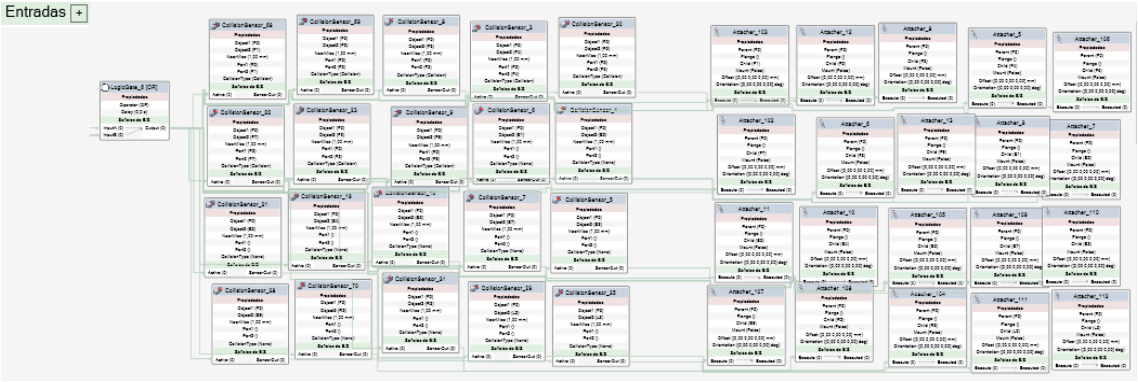


Ilustración 59. Conexión entre los sensores de colisión y los attachers del cubo en simulación

Cuando el robot se dispone a girar una cara, una señal activa el sensor de colisión de la pieza central de la misma, detectando las piezas que están junto a ella, y que, por tanto, forman la cara.

Las piezas detectadas se pegan a la central mediante los attachers, haciendo que cuando el robot agarre la cara, esta gire como una única pieza. Una vez completado el giro, una señal activa el detacher de la cara girada, haciendo que las piezas que la forman se despeguen.

En la ilustración 60 se detalla en un esquema la secuencia de acciones para el giro de una cara.

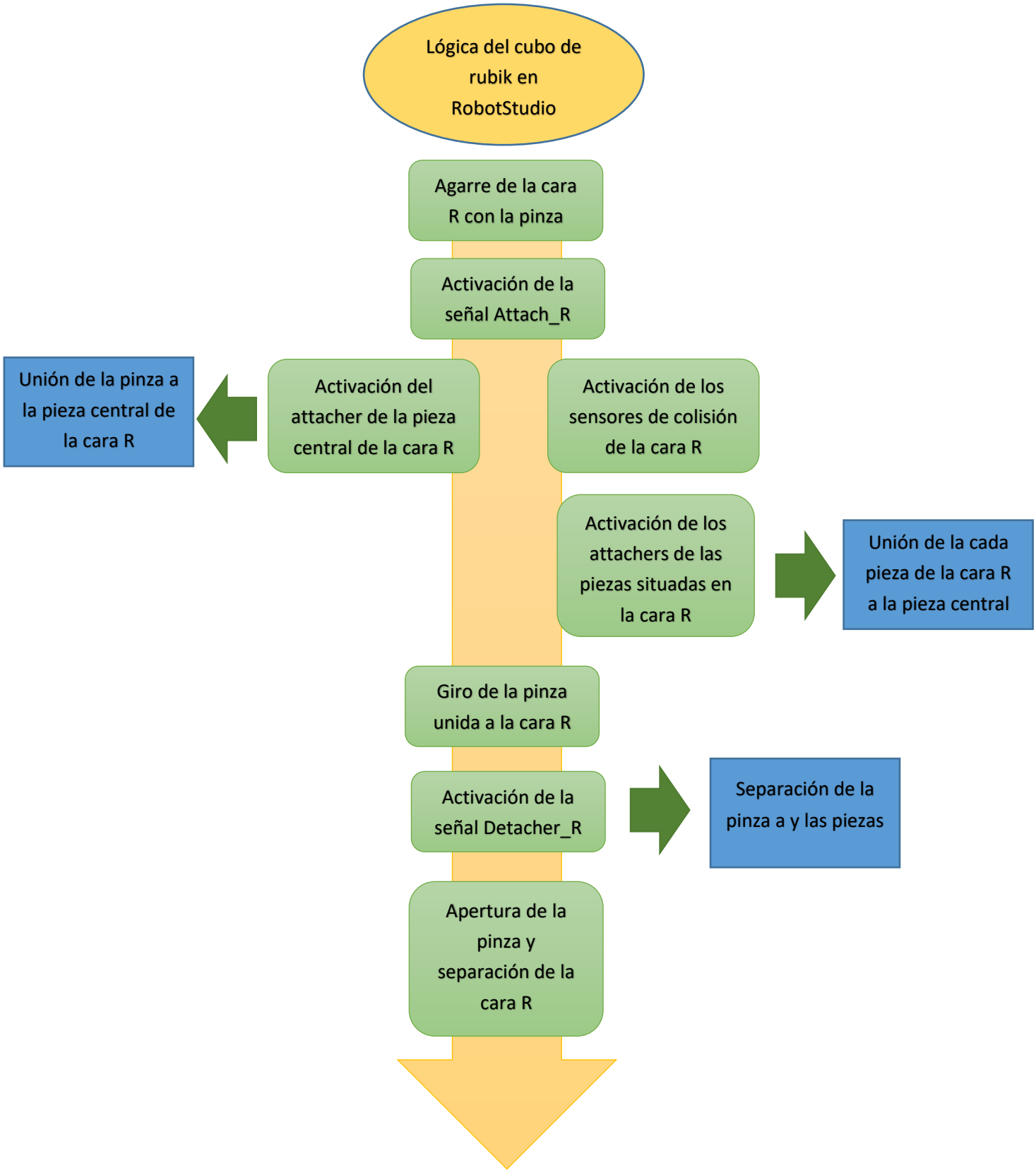


Ilustración 60. Diagrama del funcionamiento del cubo de rubik en simulación

5.3.7.3.1 Cubo 2X2

El cubo de 2x2 es similar al de 3x3, con la diferencia de que no tiene una pieza central en cada cara y, por tanto, ninguno de las ocho piezas que lo forman mantiene su posición, pudiendo pertenecer a cualquiera de las caras.

Por tanto, se han creado seis cubos, uno en el centro de cada cara que se tomarán como referencia de cada una de ellas y cumplirán las funciones de la pieza central en el cubo de 3x3. (Ilustración 61).

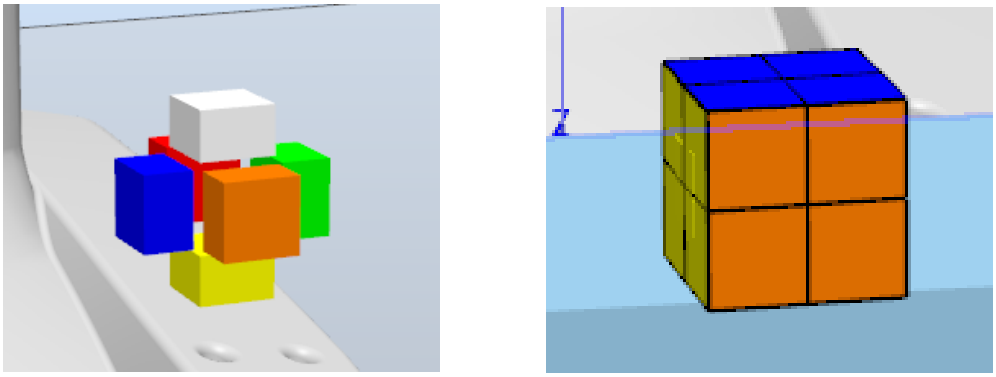


Ilustración 61. Generación del cubo 2x2 en RobotStudio

5.3.7.4 Simulación de los movimientos del robot

Una vez programados todos los componentes inteligentes y mecanismos, se puede realizar una simulación precisa de los movimientos que realizará el robot para resolver el cubo. Esto permitirá comprobar que la configuración y diseño del sistema son correctos y que el robot es capaz de alcanzar todas las posiciones deseadas y, por tanto, de resolver el cubo (2x2 o 3x3) sin problemas.

Con tal fin, RobotStudio permite hacer un análisis de colisiones activando el denominado *evitador de colisiones*. Esta herramienta, como su propio nombre indica, muestra las posibles colisiones que el robot puede sufrir durante sus movimientos, permitiendo detener la simulación y corregir las trayectorias para evitar el choque.

Para ello es necesario definir cada uno de los objetos en el sistema de evitación de colisiones. Se puede fijar la distancia de seguridad a la que se active el aviso de colisión, las tolerancias interior y exterior, y por su puesto la posición en la que se encontrará el objeto; o, en caso de ser móvil, el mecanismo al que irá sujeto.

En la ilustración 62 se muestra la herramienta ofrecida por RobotStudio para la definición de los objetos en el evitador de colisiones.

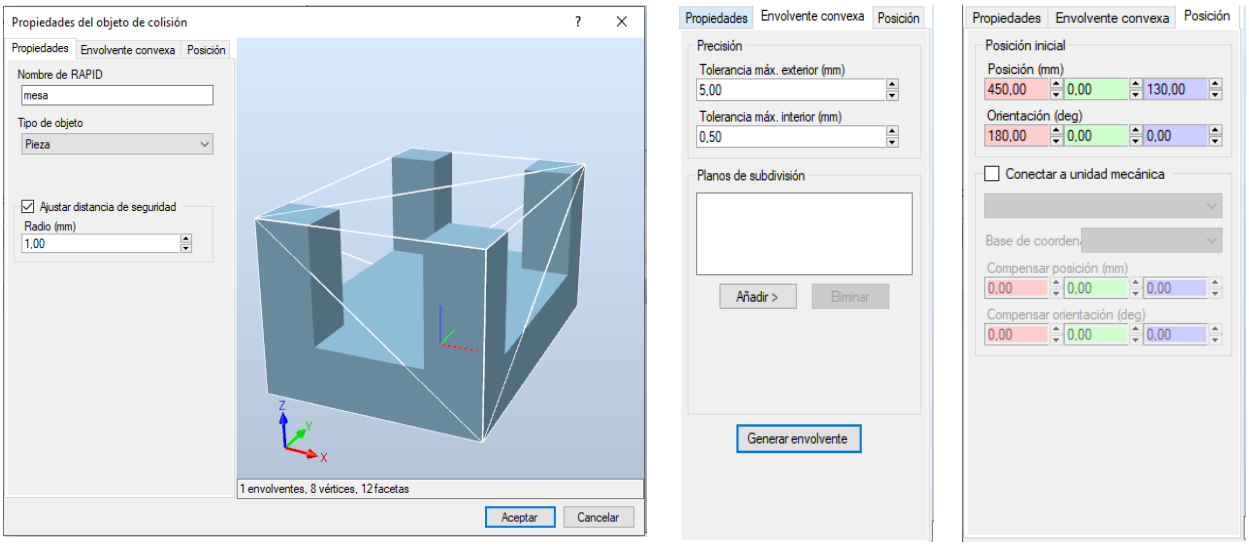


Ilustración 62. Definición de un objeto en el evitador de colisiones de RobotStudio

En este momento ya se puede comenzar a generar las trayectorias que realizará el robot para la resolución del cubo. Como es habitual, habrá una ligera discrepancia entre las trayectorias generadas en la simulación y las que ejecutará el robot real; no obstante, es lo suficientemente precisa como para validar el diseño de la estación y poder comenzar con la implementación.

5.3.7.5 Programación de los movimientos para la resolución del cubo

El último paso es la programación de cada uno de los movimientos que el robot ha de realizar para resolver el cubo.

5.3.7.5.1 Sujeción del cubo

Como se ha detallado en el apartado 5.3.6, los movimientos a realizar se envían a Robotstudio en forma de array desde Matlab, a través de una conexión TCP/IP. A continuación, y tras asegurar que ambos brazos robóticos disponen de los datos, se crea una señal de sincronización para que comiencen a procesarlos a la vez. A partir de aquí, en función de que cara haya que girar, los brazos realizarán los movimientos correspondientes.

Para la resolución del cubo se disponen de 18 giros posibles, tres por cara. Sin embargo, el hecho de girar una cara en sentido horario, anti horario o dos veces apenas influye en la trayectoria del robot, ya que el único eje que varía es el sexto, cuya rotación únicamente afecta a la orientación de la pinza.

Una de las partes fundamentales de la resolución del cubo son los agarres. Es imprescindible que el cubo esté sujeto de forma correcta y segura en todo momento por al menos uno de los brazos, ya que, si el cubo se cae o se gira en exceso, el robot lo detectaría y detendría la resolución.

Para que esto no ocurra, es vital una buena sincronización entre los brazos y una buena elección de los agarres. En total hay 24 agarres posibles, puesto que cada brazo puede agarrar cada una de las caras en horizontal o en vertical. No obstante, tras analizar cada tipo de agarre se ha llegado a la conclusión de que la solución más eficiente es a la vez la más sencilla, es decir, utilizar únicamente dos agarres.

El brazo derecho agarra la cara derecha (R) en horizontal, mientras que el izquierdo hará lo propio con la cara izquierda (L) en vertical. De esta forma, ambos brazos pueden agarrar el cubo a la vez, haciendo que los cambios de agarre sean seguros y sencillos (Ilustración 63).

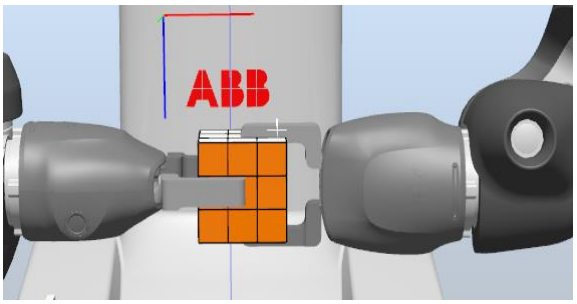


Ilustración 63. Agarres del cubo de rubik de 3x3

Al agarrar el cubo con el brazo derecho, las caras de arriba (U), abajo (D) e izquierda (L) quedan libres para que el brazo izquierdo pueda girarlas, mientras que, al agarrarlo con el brazo izquierdo, las que quedan libres son la frontal (F), la trasera (B) y la derecha (R). De este modo, únicamente con dos agarres es posible girar todas las caras.

Evidentemente, con esta configuración se necesitan más cambios de agarres que con otras en las que se programa un mayor número de agarres. Sin embargo, los cambios son más rápidos y sencillos, resultando un consumo de tiempo menor.

5.3.7.5.2 Generación de trayectorias

A raíz de lo expuesto en el apartado anterior, cada uno de los brazos puede girar tres caras. El derecho las caras frontal, trasera y derecha, y el izquierdo la superior, inferior e izquierda. Para que un brazo pueda girar una cara, tiene que sujetar únicamente dicha cara, mientras que el otro brazo tiene que agarrar la cara opuesta y la central, tal y como se aprecia en la ilustración 64.

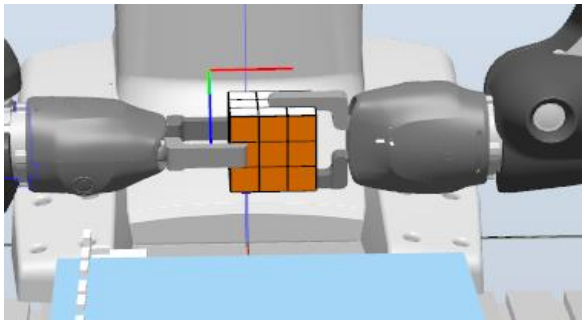


Ilustración 64. Posición de los brazos del robot para el giro de una cara del cubo

A modo de ejemplo, se desarrollarán los pasos a seguir para girar la cara derecha.

Antes de leer cada dato movimiento del array hay una tarea de sincronización que asegura que ambos movimientos inician el movimiento a la vez. Una vez sincronizados, al tratarse de una cara que debe de girar el brazo derecho, el brazo izquierdo ejecuta la instrucción Agarre_L, que implica que sujeta la cara izquierda y la central.

Una vez el cubo está seguro, el brazo derecho ejecuta la rutina Girar_R, que consiste en los siguientes movimientos:

(1) Apertura de la pinza
(2) Movimiento a la posición Apro_R,
(3) Movimiento lineal hasta R1
(4) Cierre de pinza
(5) Giro de la pinza que implica el giro de la cara R
(6) Apertura de la pinza
(7) Movimiento a la posición Apro_R
(8) Llamada a rutina Agarre_R para asegurar la sujeción del cubo para el siguiente movimiento

A continuación, se muestra el código de ambos brazos (Ilustraciones 65 y 66) y una simulación del giro. (Ilustración 67).

Código del brazo izquierdo:

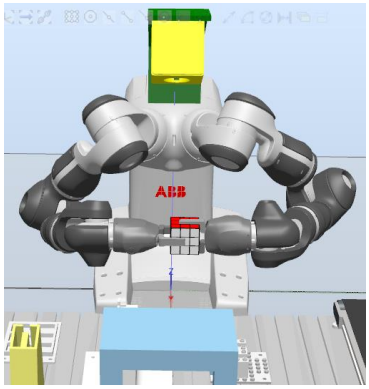
```
    ELSEIF vec{i}=20 THEN
    Agarre_L5;
    WaitSyncTask sync_RL_5,ListaTareas_RL;

PROC Agarre_L5()
    MoveL Agarre_L_1,v1000,z100,Servo\WObj:=WO_Cubo;
    Set Gripper_L;
    Reset Gripper_L;
```

Ilustración 65. Código del brazo izquierdo para el giro de la cara R

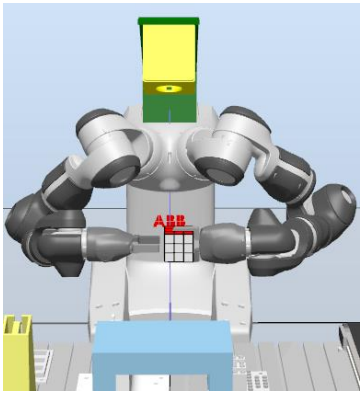
Código del brazo derecho:

```
    ELSEIF vec{i}=20 THEN
    waittime 1;
    Girar_R;
    WaitSyncTask sync_RL_5, ListaTareas_RL;

PROC Girar_R()
    Set Sensor_Pinza_R;
    MoveJ Apro_R5,v1000,z100,Servo\WObj:=wobj0;
    WaitTime 1;
    MoveL R1,v1000,z100,Servo\WObj:=wobj0;
    Set Gripper_R;
    Reset Sensor_Pinza_R;
    WaitTime 1;
    Set Attach_R5;
    MoveL R2,v1000,z100,Servo\WObj:=wobj0;
    WaitTime 1;
    Reset Attach_R5;
    Set Detach_R5;
    Set Sensor_Pinza_R;
    WaitTime 1;
    MoveL Apro_R5,v1000,z100,Servo\WObj:=wobj0;
    Reset Detach_R5;
    Reset Gripper_R;
    Reset Sensor_Pinza_R;
ENDPROC
```

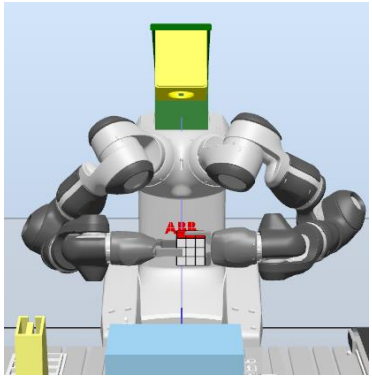
Ilustración 66. Código del brazo derecho para el giro de la cara R



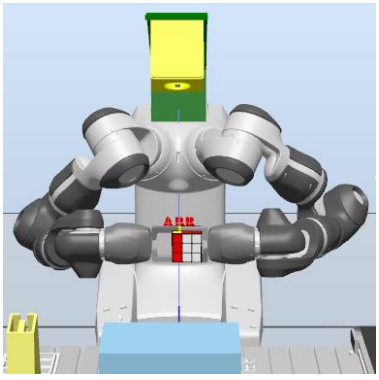
1. El brazo izquierdo se mueve a la posición de Agarre_L pasa sujerar el cubo .



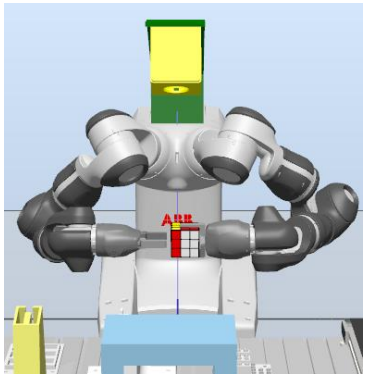
2. EL brazo derecho abre su pinza y se desplaza a la posición Apro_R2.



3. El brazo derecho se mueve a la posición R1, cerrando su pinza y sujetando la cara derecha (R) del cubo.



4. El brazo derecho gira su pinza 90º hasta la posición R2 mientras sujeta la cara R, girándola consigo.



5. EL brazo derecho abre su pinza y vuelve a la posición Apro_R donde espera al próximo movimiento

Ilustración 67. Simulación del giro de la cara R

Esta rutina es análoga para los giros de la cara trasera por parte del brazo derecho y de las caras Izquierda y superior por parte del brazo izquierdo.

Sin embargo, a la hora de girar las caras frontal e inferior, el robot es incapaz de alcanzar las posiciones para hacerlo directamente, por lo que el brazo encargado de la sujeción del cubo durante el giro realiza un giro de 90 grados en su sexto eje, es decir, en el de la pinza, girando el cubo y colocando la cara en una posición en la que el otro brazo pueda girarla.

En estos casos, el algoritmo queda de la siguiente forma:

(1) Apertura de la pinza
(2) Movimiento a la posición Apro_R,
(3) Rotación del sexto eje del brazo izquierdo, girando el cubo 90º
(4) Movimiento lineal hasta R1
(5) Cierre de pinza
(6) Giro de la pinza que implica el giro de la cara R
(7) Apertura de la pinza
(8) Movimiento a la posición Apro_R
(9) Rotación del sexto eje del brazo izquierdo, girando el cubo -90º
(10) Llamada a rutina Agarre_R para asegurar la sujeción del cubo para el siguiente movimiento

En las ilustraciones 68 y 69 se muestra el código para el giro de la cara inferior (D) y en la 70, imágenes de la simulación.

Código del brazo derecho:

```
    ELSEIF vec{i}=60 THEN
     waittime 1;
     Rotar_R;
     WaitSyncTask sync_RL_5, ListaTareas_RL;
     WaitSyncTask sync_RL_6, ListaTareas_RL;
     waittime 1;
     Rotar_R1;
     WaitSyncTask sync_RL_7, ListaTareas_RL;

PROC Rotar_R()
    Set Sensor_Pinza_R;
    WaitTime 1;
    MoveL Agarre_R,v1000,z100,Servo\WObj:=wobj0;
    Set Gripper_R;
    WaitTime 1;
    Set Rotar_Cubo_R;
    MoveL Agarre_R2,v1000,z100,Servo\WObj:=wobj0;
    WaitTime 1;
    Reset Rotar_Cubo_R;
    Set Soltar_Cubo;
    WaitTime 1;
    Reset Soltar_Cubo;
ENDPROC
PROC Rotar_R1()
    MoveL Agarre_R2,v1000,z100,Servo\WObj:=wobj0;
    WaitTime 1;
    Set Rotar_Cubo_R;
    MoveL Agarre_R,v1000,z100,Servo\WObj:=wobj0;
    WaitTime 1;
    Reset Rotar_Cubo_R;
    Set Soltar_Cubo;
    WaitTime 1;
    Reset Soltar_Cubo;
    Reset Gripper_R;
    Reset Sensor_Pinza_R;
ENDPROC
```

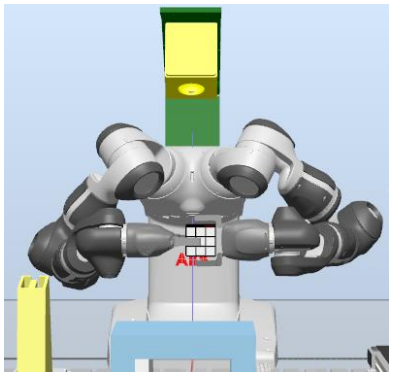
Ilustración 68. Código del brazo derecho para el giro de la cara D

Código del brazo izquierdo:

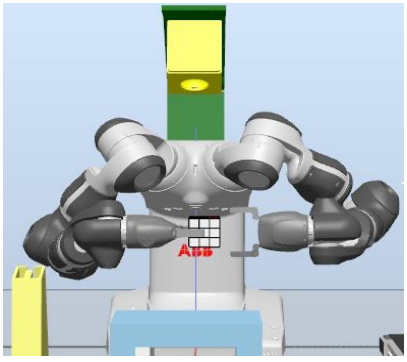
```
ELSEIF vec{i}=60 THEN
 MoveL Apro_L_1, v1000,fine,Servo;
 WaitSyncTask sync_RL_5, ListaTareas_RL;
 Girar_D;
 WaitSyncTask sync_RL_6, ListaTareas_RL;
 WaitSyncTask sync_RL_7, ListaTareas_RL;
```

```
PROC Girar_D()
    Set Sensor_Pinza_L;
    MoveJ Target_110,v1000,z100,Servo\WObj:=WO_Cubo;
    MoveJ Apro_B5,v1000,z100,Servo\WObj:=WO_Cubo;
    WaitTime 1;
    MoveL B1,v1000,z100,Servo\WObj:=WO_Cubo;
    Set Gripper_L;
    Reset Sensor_Pinza_L;
    WaitTime 1;
    Set Attach_D5;
    MoveL B2,v1000,z100,Servo\WObj:=WO_Cubo;
    Set Datos_Recibidos;
    WaitTime 1;
    Reset Attach_D5;
    Set Detach_D5;
    Set Sensor_Pinza_L;
    WaitTime 1;
    MoveL Apro_B5,v1000,z100,Servo\WObj:=WO_Cubo;
    Reset Detach_D5;
    Reset Gripper_L;
    Reset Sensor_Pinza_L;
    MoveJ Target_110,v1000,z100,Servo\WObj:=WO_Cubo;
    MoveJ Apro_L_1,v1000,z100,Servo\WObj:=WO_Cubo;
ENDPROC
```

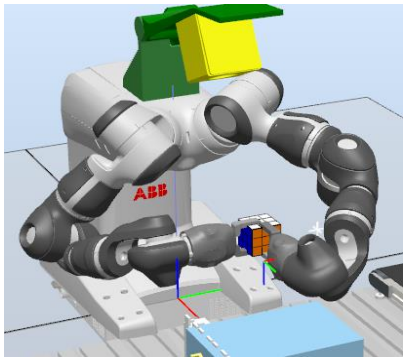
Ilustración 69. Código del brazo izquierdo para el giro de la cara D



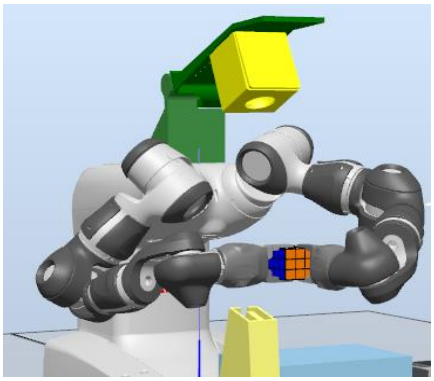
1. El brazo derecho se mueve a la posición de Agarre_R pasa sujerar el cubo .



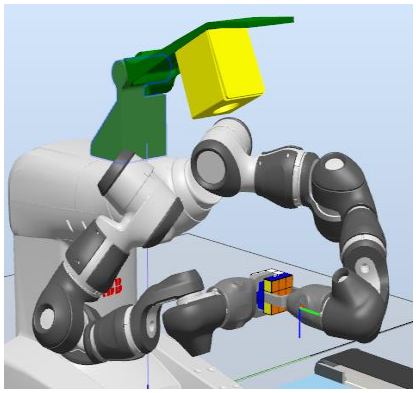
2. EL brazo derecho abre su pinza y se desplaza a la posición Apro_R.



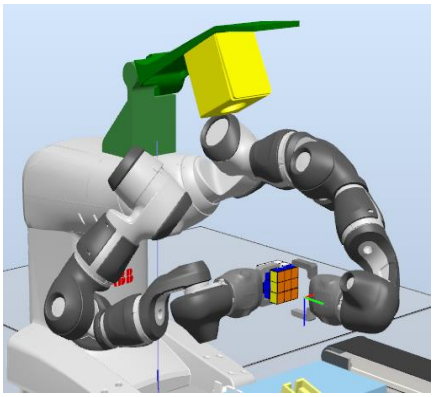
3. EL brazo izquierdo gira la pinza 90º, rotando el cubo consigo y permitiendo al brazo derecho el acceso a la cara inferior (D).



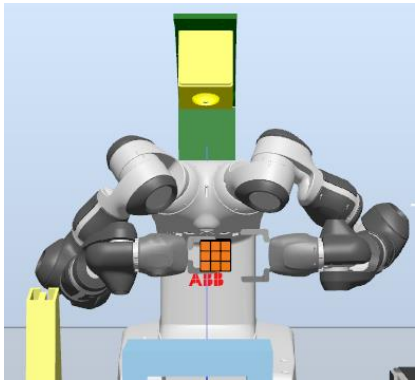
4. El brazo derecho se desplaza a B1, cerrando la pinza y agarrando la cara inferior (B).



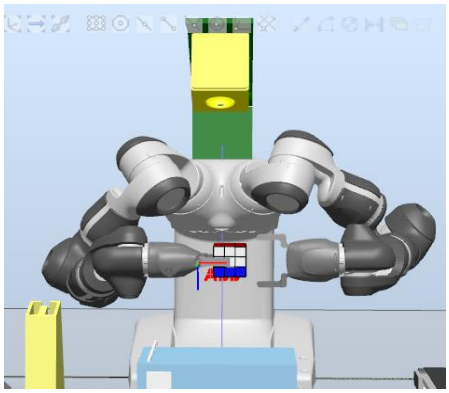
6. EL brazo izquierdo rota su pinza 90º girando la cara B con ella.



5. EL brazo derecho se mueve a la posición Apro_B.



7. EL brazo izquierdo abre su pinza y vuelve a la posición Apro_B .



8. EL brazo izquierdo vuelve a la posición Apro_L mientras que el brazo derecho rota -90º volviendo a la posición inicial.

Ilustración 70. Simulación del giro de la cara D

5.3.7.5.4 Cubo 2x2

En el caso del cubo de 2x2 la situación es diferente al no tener una fila y columna central que permita sujetar el cubo sin interferir en los giros de las filas y columnas externas. Para solventarlo se ha modificado el sistema de agarres, aumentando su número a cuatro. Dos de ellos serán los principales, el brazo derecho agarrará la cara derecha en horizontal por su parte inferior y el izquierdo la cara izquierda de forma vertical por la columna más cercana al robot. (Ilustración 71).

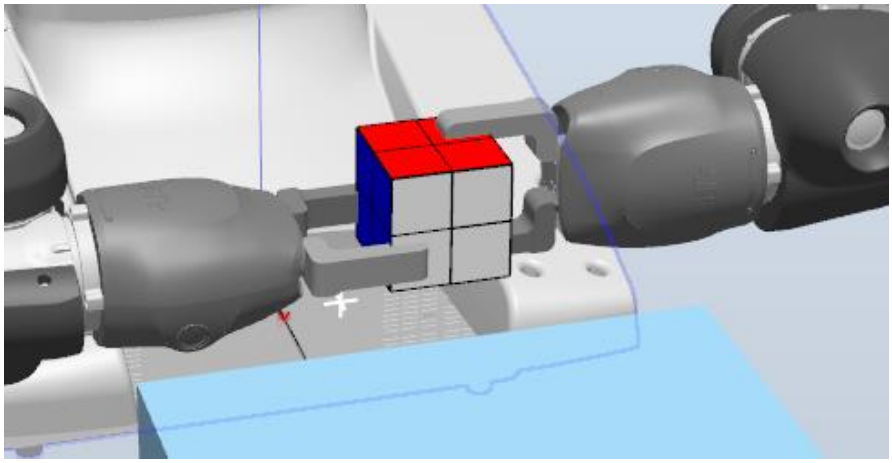


Ilustración 71. Agarres principales para el cubo de 2x2

Por otro lado, los agarres secundarios los hará el brazo izquierdo agarrando el cubo por la cara izquierda de la columna más alejada del robot y el brazo derecho agarrando la cara derecha por su columna más cercana al robot. (Ilustración 72).

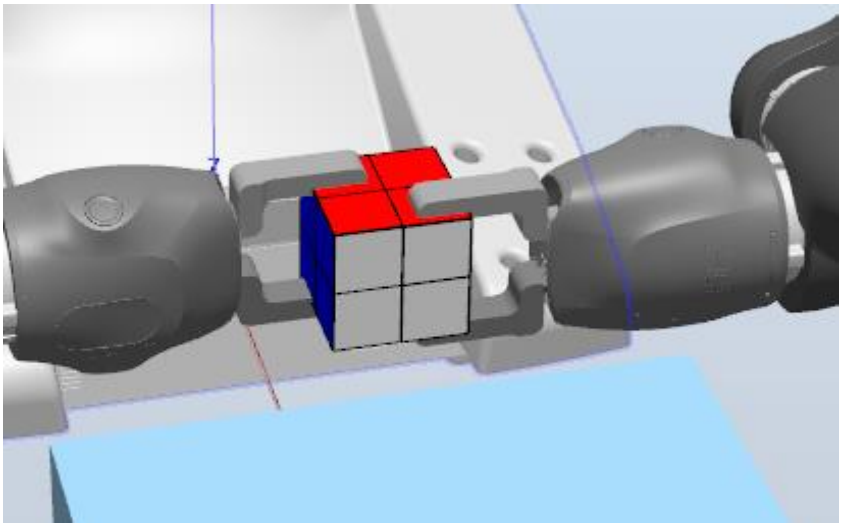
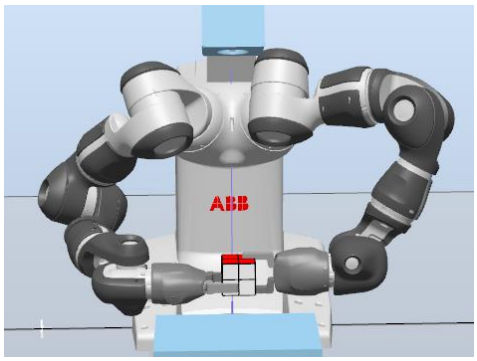


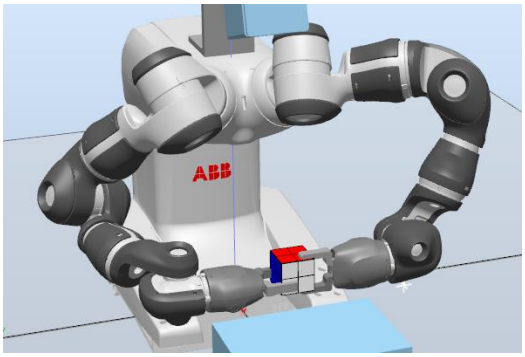
Ilustración 72. Agarres secundarios para el cubo de 2x2

Las trayectorias para los giros del cubo también varían. Para los giros de las caras derecha, izquierda, superior e inferior las trayectorias son similares. El brazo que no realiza el giro sujeta el cubo mientras que el otro brazo gira la cara deseada. El brazo derecho se encargará de los movimientos de la cara derecha y frontal mientras que el izquierdo de la superior e izquierda.

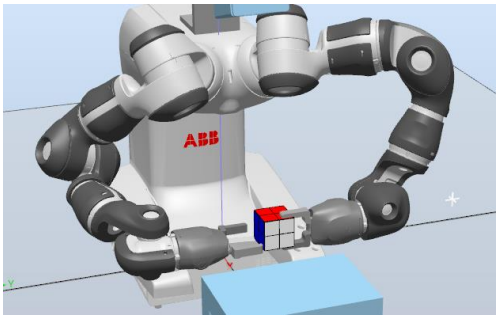
Como ejemplo en la ilustración 73 se muestra el giro de la cara frontal.



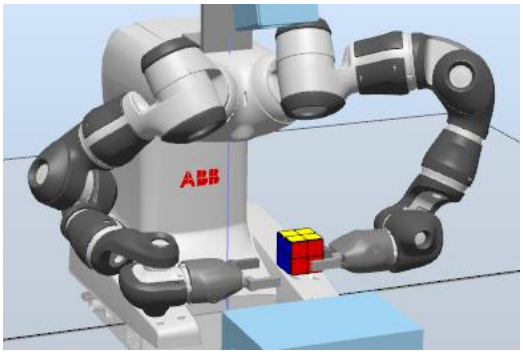
0. Partimos de la posición más habitual. El agarre principal.



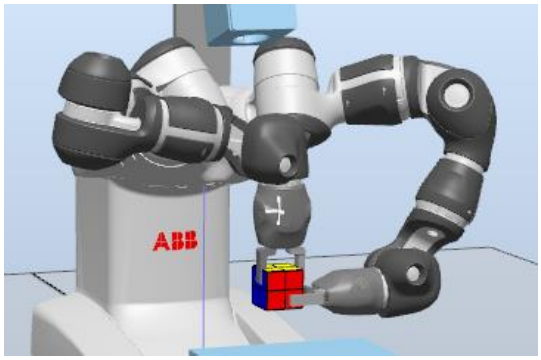
2. El brazo izquierdo cambia al agarre secundario.



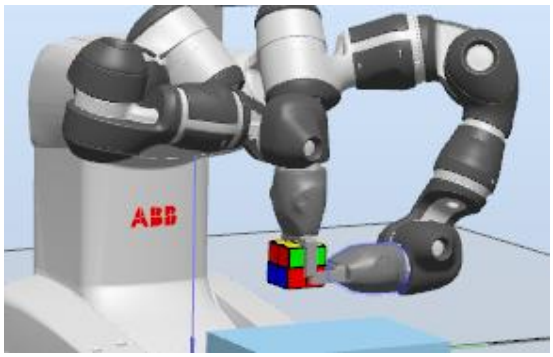
3. El brazo derecho abre la pinza y se aleja del cubo.



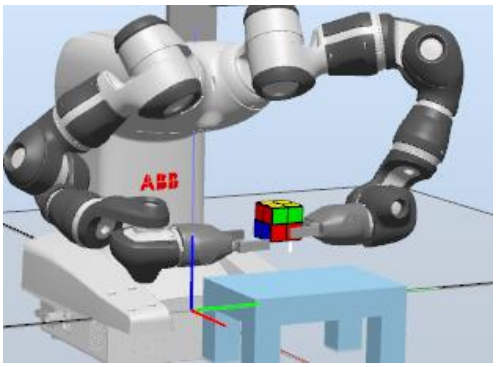
4. El brazo izquierdo gira su pinza 90º, rotando el cubo consigo y permitiendo al derecho acceder a la cara frontal



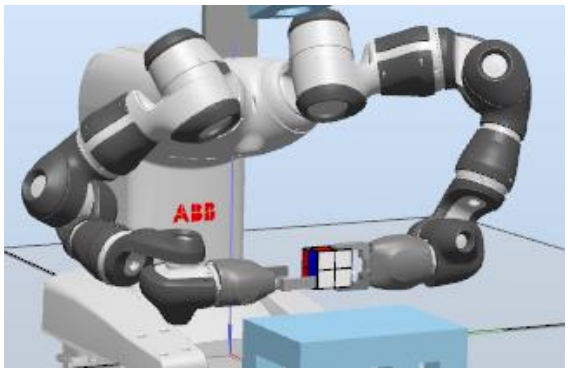
5. El brazo derecho se mueve a la cara frontal, agarrándola al cerrar su pinza.



6. El brazo derecho rota 90º, girando la cara frontal consigo



7. El brazo derecho abre la pinza y se desplaza a la posición inicial.



8. El brazo izquierdo rota su pinza -90º volviendo a la posición inicial donde ambos brazos esperan el siguiente movimiento.

Ilustración 73. Simulación del giro de la Cara F en el cubo de 2x2

Por el contrario, para los giros correspondientes a las caras inferior y trasera, y a diferencia del cubo de 3x3 donde está previamente definido que brazo gira cada cara, en este caso depende de la posición en la que finalizan el movimiento anterior.

En el caso de la cara trasera, si el giro previo ha sido en la cara frontal o izquierda, el brazo derecho agarrará el cubo utilizando el agarre secundario mientras que el brazo izquierdo realiza el giro. Si el movimiento anterior ha sido cualquier otro, el brazo izquierdo agarrará el cubo mientras que el derecho realiza el giro.

Respecto a los giros de la cara inferior, si el movimiento anterior ha sido el giro de la cara izquierda o superior, el brazo izquierdo rotará el cubo para habilitar el acceso al derecho y que pueda girarlo. Por el contrario, si el movimiento es diferente a los mencionados, será el brazo derecho quien rote el cubo para habilitar el acceso al brazo izquierdo que realizará el movimiento.

5.3.8 Programación del robot real

El último paso del proyecto es la programación del robot real Yumi. Para ello, partiremos del código generado para la simulación del RobotStudio, ya que a excepción de varias modificaciones que se comentarán a continuación, el código es completamente válido para el robot real.

Para cargar el código es necesario conectar el ordenador al puerto de servicio del Yumi a través de un cable de ethernet RJ45. Dicho puerto está configurado con la IP fija 192.168.125.1 que es la misma en todas las controladoras de ABB. Además, tiene un servidor DCHP que asigna una IP al ordenador conectado. Esta configuración permite una conexión rápida y sencilla.

En la ilustración 74 se muestra un esquema de los conectores del controlador.

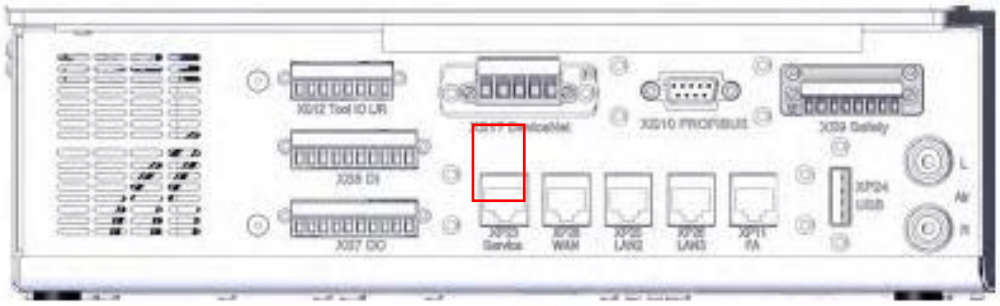


Ilustración 74. Puertos para las conexiones del robot

A partir de aquí, el ordenador detectará el controlador del robot y podrá accederse a él seleccionando la conexión en un clic. (Ilustración 75).

Ilustración 75. Conexión entre el robot y RobotStudio

Con la conexión establecida es posible cargar un nuevo código al robot o modificar el existente, e incluso controlar el robot desde el ordenador accediendo directamente a la consola FlexPendant, por lo que se podrá comenzar con la adecuación del código.

En primer lugar, las señales correspondientes a los componentes inteligentes y mecanismos creados para dotar de un mayor realismo a la simulación, como es el caso del cubo de rubik, no serán necesarios, puesto que se sustituirá por un cubo real.

Algo similar pasa con las señales de las pinzas, ya que el robot cuenta con comandos específicos para abrirlas y cerrarlas, por lo que hay que sustituir el código correspondiente a la activación y desactivación de los PoseMovers por los comandos pertinentes.

Asimismo, para que las pinzas funcionen debidamente es necesario inicializarlas y definir varias propiedades como la velocidad de cierre y apertura, la fuerza ejercida o el rango de apertura. En caso de no hacerlo tomará las propiedades predeterminadas lo que puede acarrear rupturas de los dedos si la fuerza o velocidades aplicadas son excesivas.

En la ilustración 76 se muestra el código rapid para la inicialización, apertura y cierre de las pinzas.

```
PROC Inicializacion_Pinza_L()
    g_Init \maxSpd:=25 \holdForce:=20 \Calibrate; ! Calibracion de la pinza, indicación de la fuerza máxima en N y velocidad máxima en mm/s
    WaitTime 1;
ENDPROC

  PROC Abrir_Pinza_L()
      g_GripOut \holdForce:=10; !Apertura de la pinza definiendo la fuerza ejercida
      WaitTime 0.1;
  ENDPROC

  PROC Cerrar_Pinza_L()
      g_GripIn \holdForce:=10; !Cierre de la pinza definiendo la fuerza ejercida
      WaitTime 0.1;
  ENDPROC
```

Ilustración 76. Código rapid para la inicialización, apertura y cierre de las pinzas

Cabe destacar que, en ambos casos, en caso de no eliminar dichas partes del código, el robot simplemente las ignorará. Por ello se recomienda eliminarlas o comentarlas para evitar pérdidas de tiempo innecesarias.

Aparte de la configuración de las pinzas es indispensable hacer saber al robot la forma y masa de las cargas que va a soportar, ya que de no hacerlo podría generar errores en el cálculo de las trayectorias y fuerzas de los ejes que generen graves problemas en el robot.

Este proceso consiste en tres pasos. En primer lugar, hay que definir una carga con una masa y centro de gravedad cualesquiera en el código del programa.

PERS loaddata pieza1 := [0.001, [0, 0, 0.001],[1, 0, 0, 0], 0, 0, 0];

A continuación, se agarra la pieza con la pinza de la forma en la que lo hará el robot durante la ejecución del programa y se lleva a la posición óptima para la medida de cargas, que puede observarse en la ilustración 77.

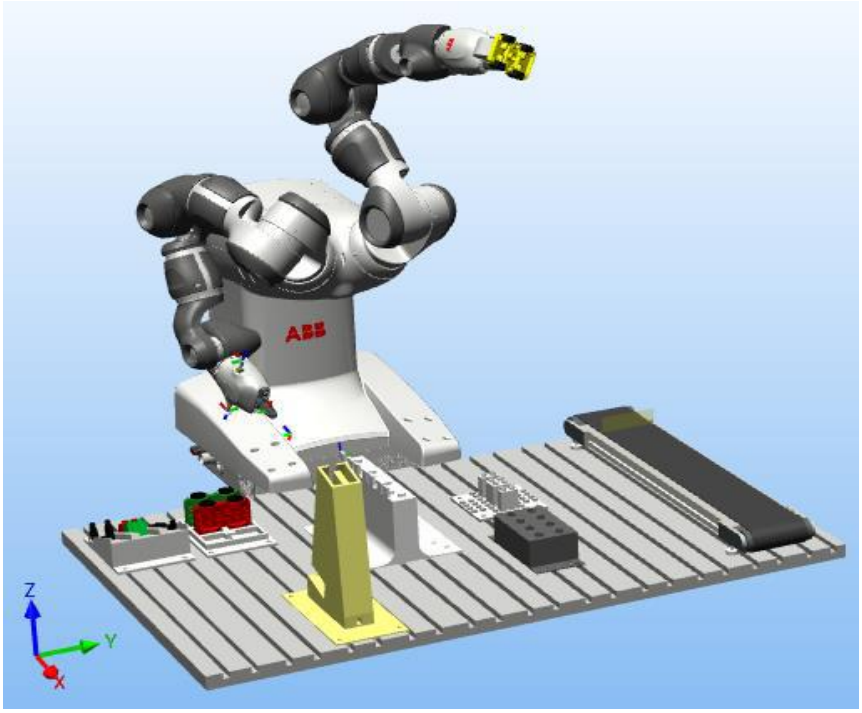


Ilustración 77. Posición óptima del brazo izquierdo para el pesaje de cargas

Con el robot en posición se ejecuta la rutina para la identificación de cargas en el FlexPendant. Este proceso hay que realizarlo para cada una de las cargas que tome el robot. Además, se debe de incluir el comando *GripLoad* pieza1 cada vez que el robot vaya a tomar la carga y *GripLoad* load0 cada vez que la suelte, de forma que sea capaz de calcular la fuerza que ejercer en cada eje para realizar los movimientos programados.

Una vez hecho esto el robot estará listo para ejecutar el programa. No obstante, como se ha explicado en el apartado anterior, es posible que existan discrepancias entre las posiciones generadas en simulación y las reales por lo que es conveniente realizar un ajuste fino de cada punto.

Para ello, mediante el FlexPendant permite, siempre y cuando el robot esté en modo manual, desplazar los brazos robóticos y modificar la posición de los puntos guardados. Por tanto, simplemente hay que ejecutar cada una de las trayectorias y comprobar que los puntos por los que pasa el robot son correctos. De no ser así, se lleva al robot a la posición deseada y se guarda en nuevo punto.

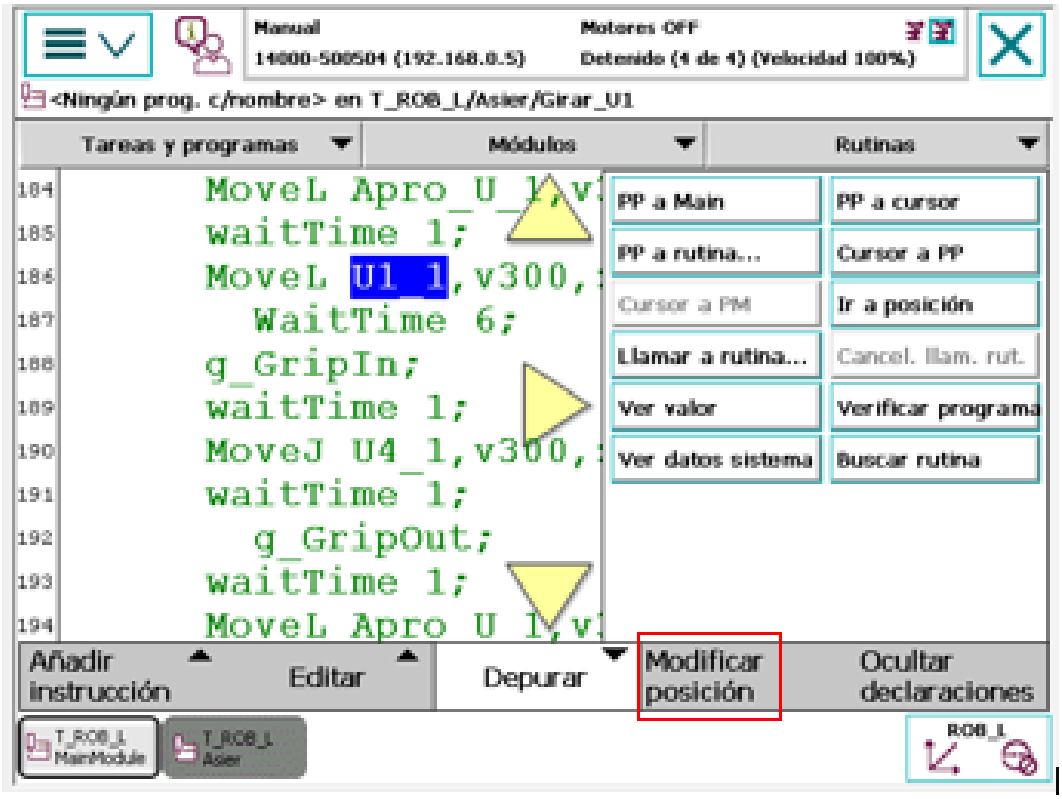


Ilustración 78. Comando del FlexPendant para la modificación de posiciones del robot

Una vez hecho esto, la programación del robot estará lista.

5.3.9 Implementación de elementos auxiliares

A parte de los elementos principales expuestos en los apartados previos, para que el sistema funcione es necesario complementarlos con elementos auxiliares. En este apartado se describirán dichos elementos para finalmente mostrar una visión completa del sistema.

Como se ha comentado en repetidas ocasiones, este proyecto se debe de implantar junto a una célula de montaje desarrollada previamente, por lo que se adaptarán parte de los elementos utilizados en ella para facilitar la implantación conjunta.

5.3.9.1 Panel de control

Ejemplo de esto es el panel de control que puede observarse en la ilustración 79, que permite la interacción entre el operario y el robot. El panel está compuesto por una caja de ABS de 190x110x90 mm del fabricante CAMDENBOSS en la que alberga la siguiente instrumentación:

- 1 Selector/switch de 2 posiciones del fabricante ABB, cuya función es la de seleccionar entre los dos procesos de ensamblaje disponibles.

- 1 Selector/switch de 3 posiciones del fabricante ABB. En caso de seleccionarse el ensamblaje de eslabones, este selector permite elegir el número de eslabones a enamblar, siendo las opciones 2, 3 o 4 eslabones.

- 2 Pulsadores RS PRO, verde y negro, que permite iniciar/reanudar y parar la ejecución del programa respectivamente.

- 2 Indicador incandescente de ARCOLECTRIC, verde y rojo. El led verde permanecerá encendido cuando todo funcione correctamente, mientras que el rojo indicará que hay algún problema en la célula.

- 1 Seta de emergencia de 2 polos del fabricante IDEC. Esta seta se conectará a la existente en el FlexPendant, permitiendo realizar una parada de emergencia desde el panel de control si fuera necesario.

Ilustración 79. Controlador de la estación

Para acoplar nuestro proyecto a los existentes, se ha modificado la programación de los dos *switches*. Debido a que al seleccionar el ensamblaje de coches de lego el segundo *switch* queda en desuso se ha generado una combinación tal que cuando se seleccione el ensamblaje de coches (*switch* ON) y cuatro eslabones, el programa que se ejecute sea la resolución el cubo de rubik. (Ilustración 80).

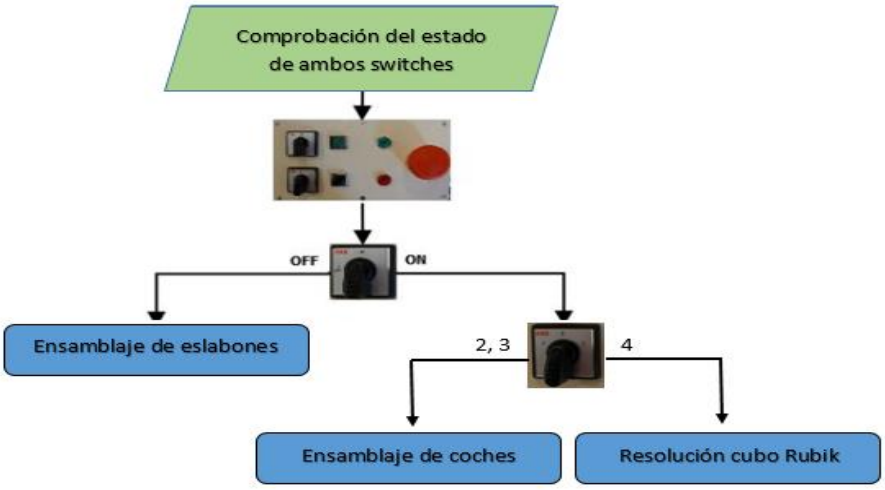


Ilustración 80. Diagrama de la selección del programa a ejecutar por el robot

5.3.9.2 Cuadro eléctrico

Para que la gran mayoría de los elementos que conforman la célula funcionen es necesario alimentarlos mediante una fuente de tensión. Al igual que ocurría en el apartado anterior, se ha aprovechado el cuadro electrico desarrollado en años anteriores añadido los componentes que fuesen necesarios.

El cuadro eléctrico existente consistía estaba dividido en tres zonas de diferentes tensiones:

Por un lado la zona de 24V a la que está conectada la alimentación del robot y todas las entradas y salidas digitales del tobot, así como la toma de tierra. En total lo conforman cuatro terminales de ocho bornes cada uno.

Por otro lado, la zona de 15V que alimenta a los componentes relacionados con la cinta transportadora encargada de surtir de piezas a la zona de ensamblaje. A ella se conectan el motor paso a paso de la cinta, el driver de micropaso que la controla y un arduino YUN.

Por último, un arduino UNO con el shield de ethernet utilizados en el ensamblaje de eslabones para el control del módulo de sonido.

A estos elementos se ha añadido una fuente de alimentación Euroconnex de 24V y 2.5A que alimenta a la cámara y al domo domo de iluminación led.

En la ilustración 81 se puede apreciar el cuadro eléctrico.

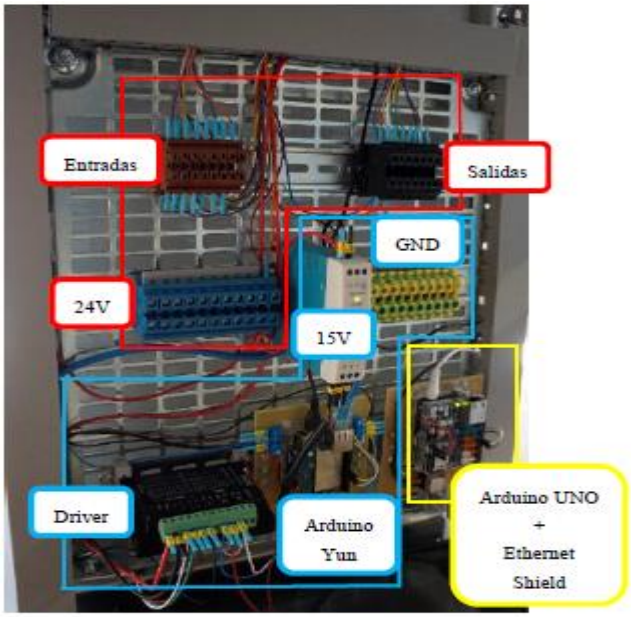


Ilustración 81. Cuadro eléctrico

Una vez conectados todos los componentes, el sistema estará completo y listo para funcionar.

Para finalizar, y a modo de resumen, en la ilustración 82 se muestra un diagrama de todo el proceso de resolución del cubo.

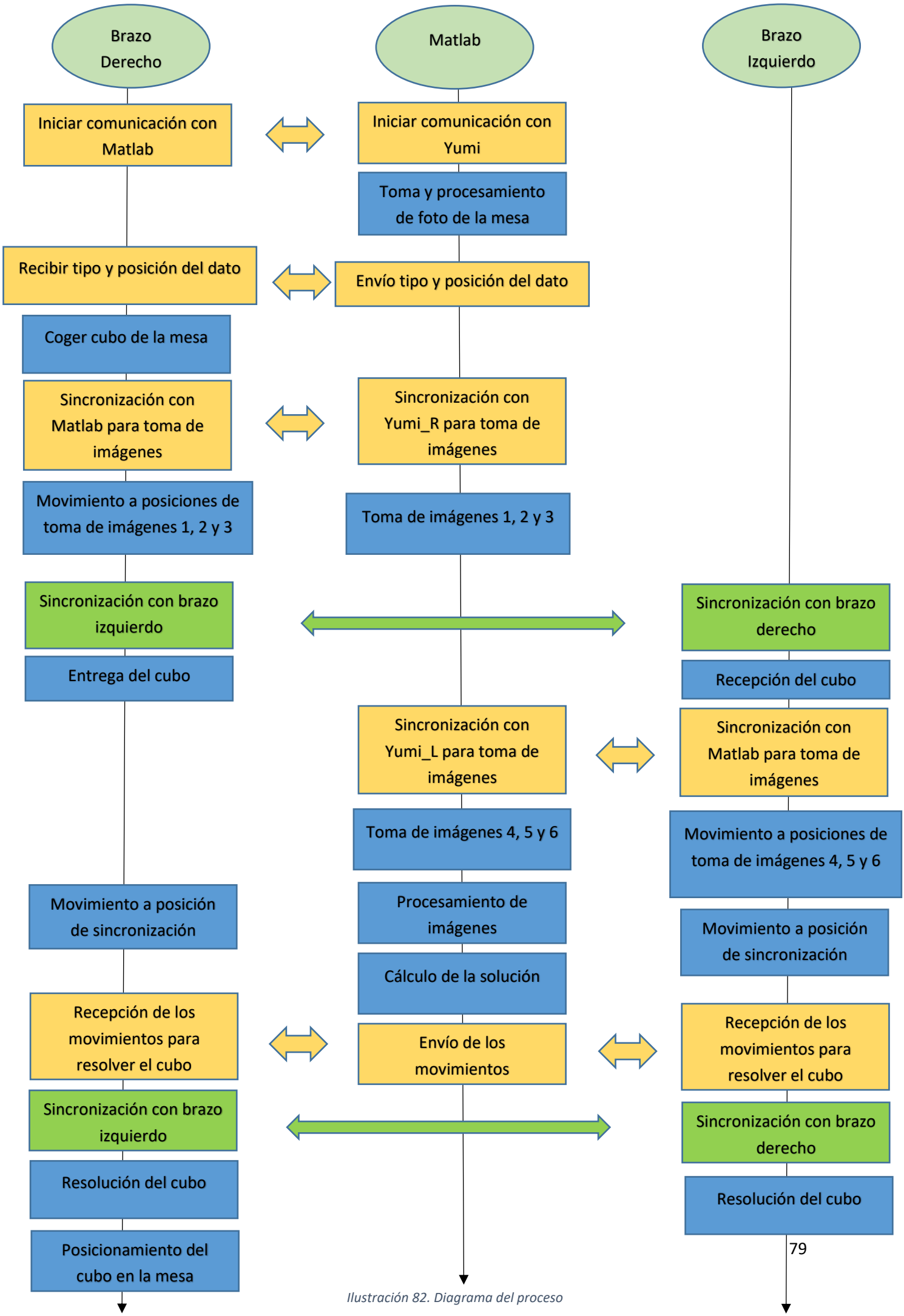


Ilustración 82. Diagrama del proceso

6. METODOLOGÍA SEGUIDA EN EL DESARROLLO DEL TRABAJO

A lo largo de este apartado se van a describir las tareas que conforman el proyecto, así como su duración y recursos materiales y humanos que han requerido.

Inicio del proyecto:

El proyecto comienza el día 27 de enero de 2020 con una reunión con la tutora del proyecto en la que se explica grosso modo en lo que va a consistir y los distintos objetivos que se deben de cumplir.

Búsqueda de información:

La primera fase se basa en la búsqueda de información sobre el desarrollo actual de las tecnologías que toman parte en proyecto. Los conocimientos adquiridos permiten plantear y elegir entre las diferentes alternativas las que mejor se adecuen a las necesidades del proyecto.

- Duración: 3 semanas
- Medios humanos: Ingeniero junior y responsable de proyecto
- Medios materiales: un ordenador

Diseño de la estación:

Con la información básica obtenida se puede comenzar a plantear la solución del proyecto. El primer paso es el diseño del sistema, es decir, seleccionar los elementos que lo van a conformar y su distribución en la estación para lograr la solución más eficiente.

- Duración: 2 semanas
- Medios humanos: Ingeniero junior y responsable de proyecto
- Medios materiales: un ordenador

Selección de los componentes:

Se buscarán en el mercado los componentes que mejor se ajusten a las necesidades del sistema y que han sido definidos en el apartado anterior. El elemento más crítico es el robot ya que tiene un papel fundamental en el proyecto y es el de más valor. Sin embargo, también se deben seleccionar componentes auxiliares como fuentes de alimentación, tarjetas de red, etc. Que son imprescindibles para el funcionamiento del sistema.

- Duración: 2 semanas
- Medios humanos: Ingeniero junior y responsable de proyecto
- Medios materiales: un ordenador

Diseño de componentes CAD:

La última fase del diseño es el modelado 3D de la carcasa y amarre de la cámara y de los dedos de las pinzas que permitirán tanto la fijación de la cámara como un correcto agarre del cubo por parte del robot.

- Duración: 2 semanas
- Medios humanos: Ingeniero junior y responsable de proyecto
- Medios materiales: Un ordenador, SW Online Onshape

Aprendizaje de Matlab y Robotstudio:

Una vez seleccionados los componentes que formarán el sistema, es necesario realizar un aprendizaje de los programas que se van a utilizar para controlar cada uno de ellos. De esta forma se les podrá sacar un mejor rendimiento y se obtendrá más eficiente.

- Duración: 3 semanas
- Medios humanos: Ingeniero junior y responsable de proyecto
- Medios materiales: un ordenador, SW Matlab, SW RobotStudio

Impresión de las piezas 3D y montaje del sistema:

Las tres semanas dedicadas al aprendizaje del uso de los programas se utilizaron paralelamente para la impresión de las piezas 3D. De esta forma, una vez las piezas están listas se puede proceder al montaje de la cámara y de las pinzas, con sus correspondientes componentes auxiliares, dejando todo preparado para la implementación del sistema.

- Duración: 3 semanas
- Medios humanos: Ingeniero junior, Técnico en impresión 3D y responsable de proyecto
- Medios materiales: SW de impresión 3D, impresora 3D y herramientas de montaje
-

Configuración de la cámara:

Con la cámara fijada en su posición se procede a la configuración de la misma, fijando las propiedades y parámetros óptimos de las fotografías para las necesidades del proyecto.

- Duración: 1 semanas
- Medios humanos: Ingeniero junior y responsable de proyecto
- Medios materiales: un ordenador, SW Matlab, SW Ueye Cockpit, SW IDS Camera Manager, Cámara

Procesamiento de imágenes:

Para surtir al robot y al algoritmo de la información necesaria para resolver al cubo es necesario realizar el procesamiento de las fotografías que toma la cámara.

- Duración: 3 semanas
- Medios humanos: Ingeniero junior y responsable de proyecto

- Medios materiales: un ordenador, SW Matlab, Cámara

Programación del algoritmo del cubo de rubik:

Partiendo de la información obtenida por el procesamiento de imágenes, se programa un algoritmo capaz de resolver el cubo de rubik de 3x3 y 2x2 en el menor número de movimientos posible.

- Duración: 3 semanas
- Medios humanos: Ingeniero junior y responsable de proyecto
- Medios materiales: un ordenador, SW Matlab

Programación de las comunicaciones entre el robot y Matlab:

Tanto para el envío de los movimientos como para sincronizar la toma de imágenes es necesario establecer una comunicación entre Matlab y el robot que permita el intercambio bidireccional de datos. En esta fase se programarán las rutinas para entablar una comunicación TCP/IP entre ambos componentes.

- Duración: 1 semana
- Medios humanos: Ingeniero junior y responsable de proyecto
- Medios materiales: un ordenador, SW Matlab, SW RobotStudio

Simulación de la estación de RobotStudio:

El último paso del proyecto es la programación del robot, para ello, en primer lugar, se genera una estación virtual en RobotStudio que permita simular los movimientos del robot y generar las trayectorias a realizar para resolver el cubo. De esta forma se podrá comprobar que las trayectorias son correctas antes de la implementación en el robot real.

- Duración: 3 semanas
- Medios humanos: Ingeniero junior y responsable de proyecto
- Medios materiales: un ordenador, RobotStudio

Programación del robot Yumi:

Partiendo del código y trayectorias generadas a partir de la simulación, se carga el programa en el robot real, realizando las modificaciones necesarias y corrigiendo las discrepancias en las posiciones entre la simulación y la estación real.

- Duración: 2 semanas
- Medios humanos: Ingeniero junior y responsable de proyecto
- Medios materiales: un ordenador, Robot Yumi, SW RobotStudio

Pruebas y optimización del sistema:

Una vez el sistema está completo se realizan las pruebas que garanticen su correcto funcionamiento y su seguridad. Además, se aprovecharán las últimas pruebas para detectar posibles mejoras que permitan optimizar el sistema.

- Duración: 3 semanas
- Medios humanos: Ingeniero junior y responsable de proyecto
- Medios materiales: un ordenador, SW Matlab, SW RobotStudio

Fin del proyecto:

Tras las pruebas y validación, se da por concluido el proyecto el 21 de septiembre de 2020.

	ACTIVIDAD	INICIO	DURACIÓN (Semanas)
H1	**Inicio del proyecto**	27-ene	0
T1	**Búsqueda de información**	27-ene	3
T2	**Diseño del sistema**	17-feb	6
T2.1	Diseño de la célula	17-feb	2
T2.2	Selección de componentes	02-mar	2
T2.3	Diseño de piezas en CAD	16-mar	2
T3	**Aprendizaje del SW**	30-mar	3
T3.1	Matlab	30-mar.	1
T3.2	RobotStudio	06-abr	2
T4	**Impresión piezas 3D y montaje del sistema**	30-mar	3
T5	**Matlab**	20-abr	7
T5.1	Configuración de la cámara	20-abr	1
T5.2	Procesamiento de imágenes	27-abr	2
T5.3	Programación del algoritmo del cubo de Rubik	11-may	3
T5.4	Pruebas y optimización	01-jun	1
T.6	**Programación de comunicaciones**	08-jun	1
T7	**RobotStudio**	15-jun	4
T7.1	Creación del sistema virtual	15-jun	2
T7.2	Simulación del sistema	29-jun	2
T8	**Programación del robot Yumi**	13-jul	5
T8.1	Modificación del código	13-jul	1
T8.2	Ajuste de las posiciónes	20-jul	2
T8.4	Pruebas y optimización	01-sep	2
T9	**Pruebas y validación del sistema completo**	14-sep	1
H2	**Fin del proyecto**	21-sep	1

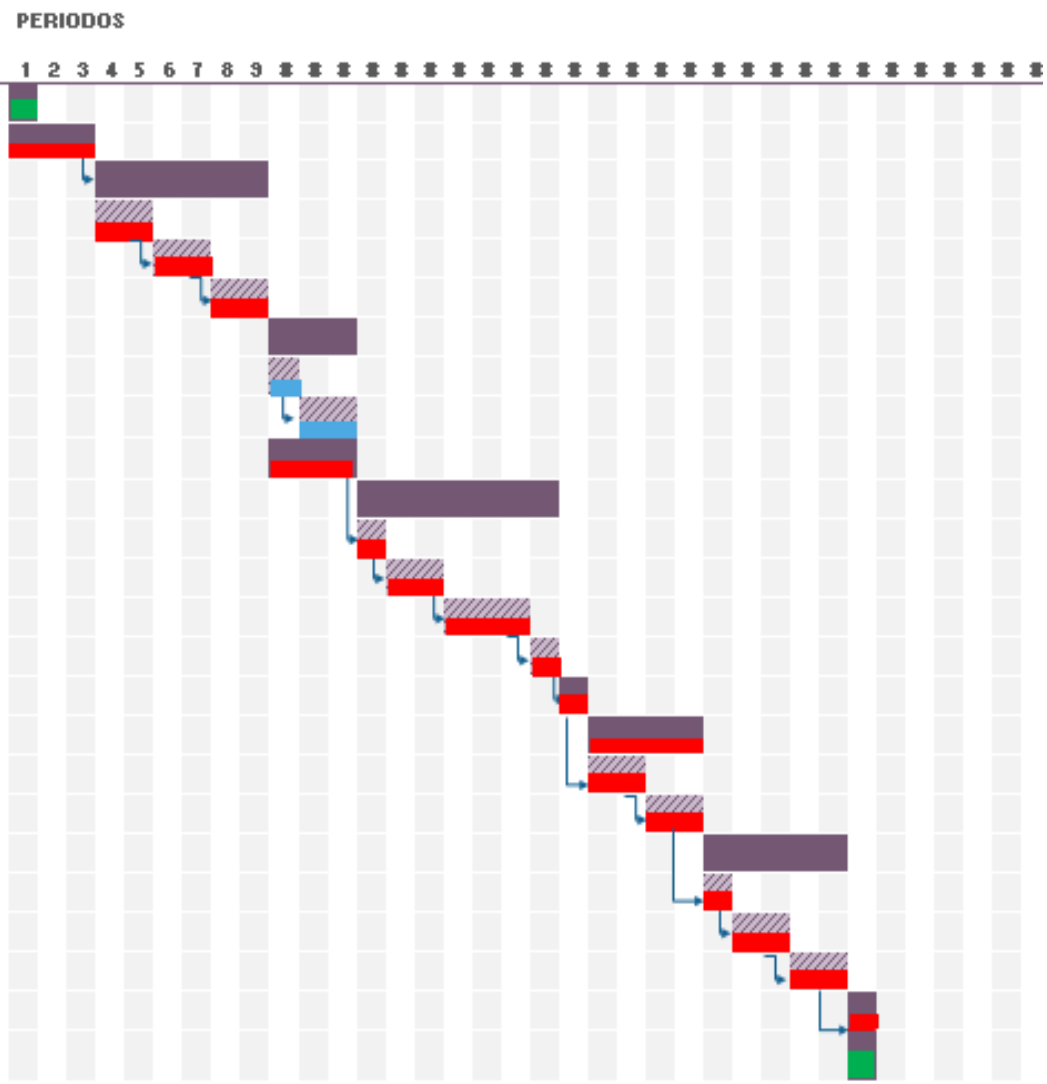


Ilustración 83. Diagrama de la planificación del proyecto

7. PRESUPUESTO

A lo largo de este apartado se realiza una aproximación de los costes que ha supuesto el proyecto.

El presupuesto se divide en dos grandes bloques, por un lado, la mano de obra y por otro las amortizaciones de los materiales y maquinaria utilizada.

Mano de obra

En el desarrollo del proyecto han tomado parte un ingeniero junior, un director del proyecto, un técnico de impresión y un técnico eléctrico. En la tabla 3 se muestran sus tasas y horas trabajadas.

	Horas (h)	**Coste unitario (€/h)**	**Coste (€)**
Ingeniero junior	600	30	18.000
Director del proyecto	50	60	3.000
Técnico de impresión	40	40	1.600
Técnico eléctrico	10	40	400
	Coste total		**23.000**

Tabla 3. Costes mano de obra

Amortizaciones

El sistema implementado en el proyecto se utilizará tanto con fines educativos como para la realización de futuros proyectos del departamento. Es por ello que, para calcular el coste vinculado a ellos se van a calcular sus amortizaciones en base a su tiempo de vida y a la duración del proyecto.

En la tabla 4 se calcula el coste que dichos elementos han supuesto al proyecto.

	Precio (€)	**Vida útil (años)**	**Tiempo de uso (meses)**	**Amortización (€)**
Robot Yumi	35.000,00	10	6	1.750,00
Cámara IDS	1581,57	8	6	93.75,00
Objetivo	200,00	8	6	12.5,00
Domo	100,00	8	6	6,25
Piezas 3D	200,00	6	6	16,67
Ordenador de sobremesa	900,00	5	2	90,00
Ordenador portátil	700,00	5	6	70,00
Mesa de trabajo	1.500,00	10	6	75,00
Cuadro eléctrico	170,00	5	6	17,00
Panel de control	150,00	5	6	150,00
MS Office	115,00	1	6	115,00
Matlab	1980,00	3	6	660,00
		Total amortizaciones		**3049.92**

Tabla 4. Coste amortizaciones

Analizando los resultados, se aprecia que la gran parte de los costes se deben al trabajo realizado por el ingeniero junior que ha desarrollado el proyecto, suponiendo un 61.5% del coste total. Por otro lado, el robot es el componente más costoso de los utilizados, siendo también la base del proyecto.

Finalmente, sumando ambos resultados se obtiene el coste total del proyecto, que asciende a **26.049,92€**

8. CONCLUSIONES

Finalmente, en este último apartado se describirán las conclusiones obtenidas a lo largo del desarrollo del proyecto.

En primer lugar, cabe mencionar que se ha cumplido el objetivo principal, lográndose implementar un sistema capaz de resolver cubos de rubik de manera autónoma.

A nivel particular caben destacar los siguientes apectos:

- El modelado e impresión de piezas 3D que complementasen a los elementros principales del sistema.

- La implementación de un sistema de visión artificial capaz de tomar y procesar imágenes. Permitiendo extraer de ellas toda la información necesaria para la resolución del cubo de rubik.

- La programación un algoritmo capaz de resolver el cubo de rubik, independientemente de su estado inicial, en un número de movimientos aceptable.

- La programación del robot colaborativo Yumi para la ejecución de los movimientos que resuelvan el cubo.

- El establecimiento de las comunicaciones entre los diferentes elementos del sistema que han permitido su cooperación y sincronización.

De esta forma, se ha comprobado que la combinación de la robótica colaborativa con la visión artificial aporta grandes beneficios, aunando en un único sistema las virtudes de la colaboración entre humanos y robots con la capacidad de recabar información de una cámara y el procesamiento de imágenes.

Además, a nivel personal, se han adquirido los conocimientos esperados tanto en las tecnologías utilizadas como en la gestión del propio proyecto, habiéndose superado las dificultades generadas por el virus Sars-Covid 19.

REFERENCIAS

[1] R. Y. Zhong, X. Xu, E. Klotz, and S. T. Newman, “Intelligent Manufacturing in the Context of Industry 4.0: A Review”, *Engineering*, vol. 3, no. 5, pp. 616–630, Oct 2017.

[2] K.-D. Thoben, S. Wiesner, and T. Wuest, ““Industrie 4.0” and Smart Manufacturing – A Review of Research Issues and Application Examples”, *International Journal of Automation Technology*, vol. 11, no 1, pp. 4–19, Jan. 2017.

[3] European Comission. The next research andinnovation framework programme, 2020. https://ec.europa.eu/info/horizon-europe-next-research-and-innovation-framework-programme_en

[4] H. Ahuett-Garza, and T.Kurfess, “A brief discussion on the trends of habilitating technologies for Industry 4.0 and Smart manufacturing”, *Manufacturing Letters*, vol. 15, pp. 60–63, 2018.

[5] Vaidya, S., Ambad, P., Bhosle, S., “Industry 4.0 – A Glimpse”. Procedia Manufacturing, vol. 20, pp. 233-238, 2018.

[6] Erboz, Gizem. “How To Define Industry 4.0: Main Pillars Of Industry 4.0” Managerial trends in the development of enterprises in globalization era. Faculty of Economics and Social Sciences, Business and Management, Hungary ,2017.

[7] G. Du, M. Chen, C. Liu, B. Zhang, and P. Zhang, “Online Robot Teaching With Natural Human–Robot Interaction”, *IEEE Transactions on Industrial Electronics*, vol. 65, no. 12, pp. 9571–9581, Dec. 2018.

[8] Universal Robots. La revolución colaborativa en las fábricas del futuro, 2019. https://blog.universal-robots.com/es/la-revolucion-colaborativa-en-las-fabricas-del-futuro

[9] E. Lee, A. Barthelmey, T. Reckelkamm, H. Kang and J. Son, "A Study on Human-Robot Collaboration based Hybrid Assembly System for Flexible Manufacturing," *IECON 2019 - 45th Annual Conference of the IEEE Industrial Electronics Society*, Lisbon, Portugal, 2019, pp. 4197-4202.

[10] E. Matheson, R. Minto, E. G. G. Zampieri, M. Faccio and G. Rosati, "Human–Robot Collaboration in Manufacturing Applications: A Review", *Robotics,* vol. 8, no. 100, 2019.

[11] T. Smith, P. Bernardos and D. Branson, "Assessing worker performance using dynamic cost functions in human-robot collaborative tasks", *Mechanical Engineering Science,* vol. 234, pp. 289-301, 2020.

[12] M. J. Rosenstrauch and J. Krüger, "Safe human-robot collaboration - Introduction and experiment using ISO/TS 15066", 3rd International Conference on Control, Automation and Robotics, Institute for Machine Tools and Factory Managements, pp. 740-744, Germany, 2017.

[13] Universal Robots. ¿Cómo mejoran los cobots la industria médica y farmacéutica? 2018 https://blog.universal-robots.com/es/cobots-industria-m%C3%A9dica-y-farmac%C3%A9utica

[14] BBVA. La era de los robots colaborativos: los cobots, 2019 https://www.bbvaopenmind.com/tecnologia/robotica/la-los-robots-colaborativos-los-cobots/#:~:text=Ejemplos%20de%20uso%20de%20Cobots,a%20medida%20para%20la%20aplicaci%C3%B3n.

[15] T. Smith, P. Bernardos and D. Branson, "Assessing worker performance using dynamic cost functions in human-robot collaborative tasks", Mechanical Engineering Science, vol. 234, pp. 289-301, 2020.

[16] Barosz, P.; Gołda, G.; Kampa, A. "Efficiency Analysis of Manufacturing Line with Industrial Robots and Human Operators", Applied Science- BASEL., vol.10, no. 8, art. 2862, Poland, 2020

[17] Interact Analysis. The collaborative robot market-2019, 2019 https://www.interactanalysis.com/the-collaborative-robot-market-2019-infographic/

[18] Marino MV, Shabat G, Gulotta G, Komorowski AL. "From Illusion to Reality: A Brief History of Robotic Surgery". *Surgical Innov*ation, vol. 25(3), pp. 291-296, 2018.

[19] Universal Robots. El hospital de Gentofte, 2020. https://www.universal-robots.com/es/casos-pr%C3%A1cticos/hospital-de-gentofte/

[20] Kuka. Robot assisted rehabilitation, 2020. https://www.kuka.com/en-us/industries/solutions-database/2019/08/robert-from-life-science-robotics

[21] Kuka. Timo Boll – The spin o flife, 2020. https://www.kuka.com/timo

[22] Toyota. The development of CUE, the AI Basketbal robot, 2020. https://global.toyota/en/newsroom/corporate/28595150.html

[23] Nao – Team HTWK, 2020.

https://www.htwk-robots.de/

[24] Ai-Da Robot. Who is Ai-Da?, 2020.

https://www.ai-darobot.com/general-interest

[25] Van Aerschot, Lina & Parviainen, Jaana. “Robots responding to care needs? A multitasking care robot pursued for 25 years, available products offer simple entertainment and instrumental assistance”, Ethics and Information Technology. 2020.

[26] P. Ponce, A. Molina, D. Grammatikou, O. Mata, “Fuzzy logic type 1 and 2 for social robots and apps for children with autism”. 17th Mexican International Conference on Artificial Intelligence – Special Session, pp. 1-8, Mexico, 2017.

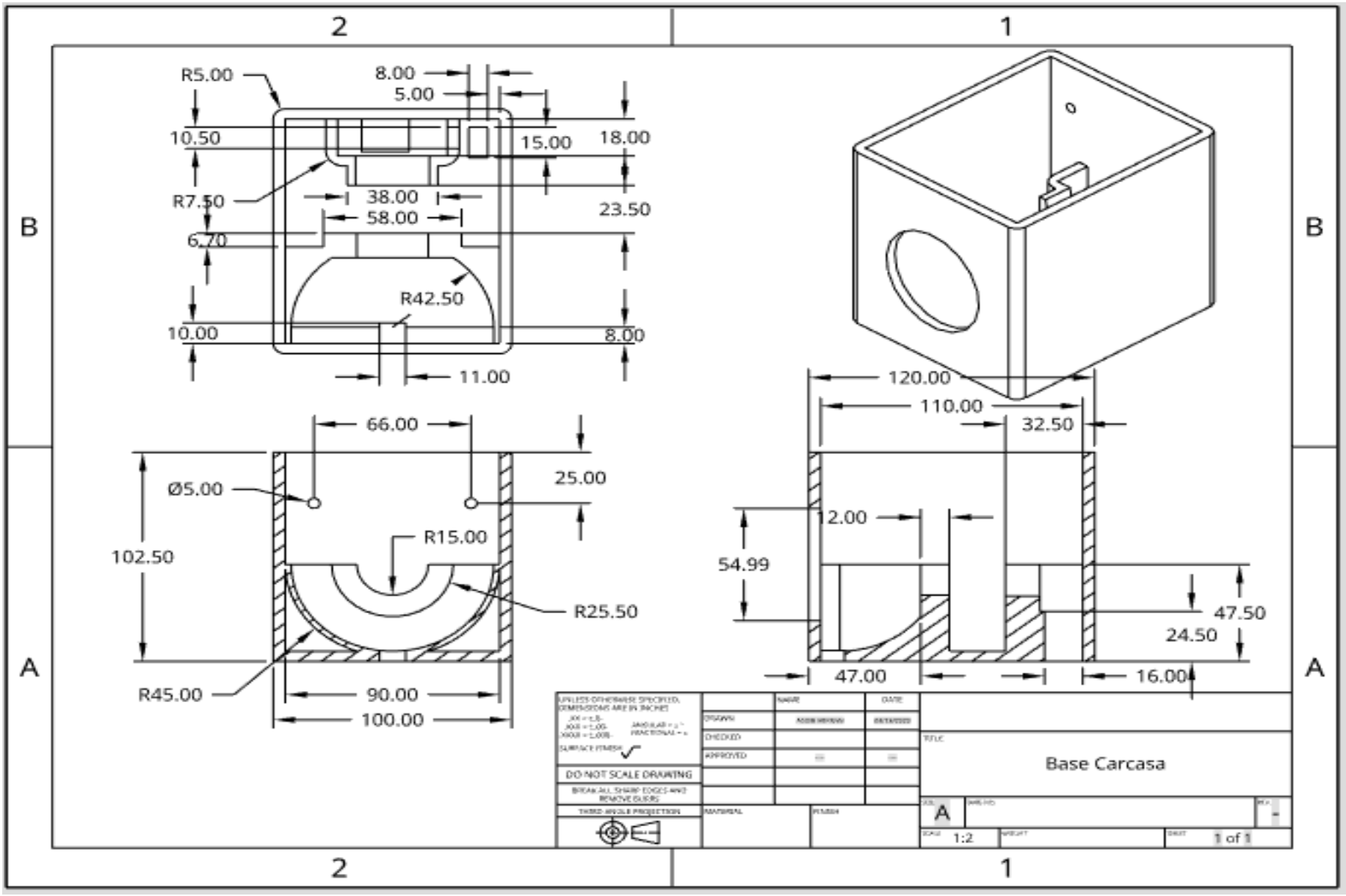
[27] S. Moharana, A. E. Panduro, H. R. Lee and L. D. Riek, "Robots for Joy, Robots for Sorrow: Community Based Robot Design for Dementia Caregivers," *14th ACM/IEEE International Conference on Human-Robot Interaction (HRI)*, Daegu, Korea (South), pp. 458-467, 2019.

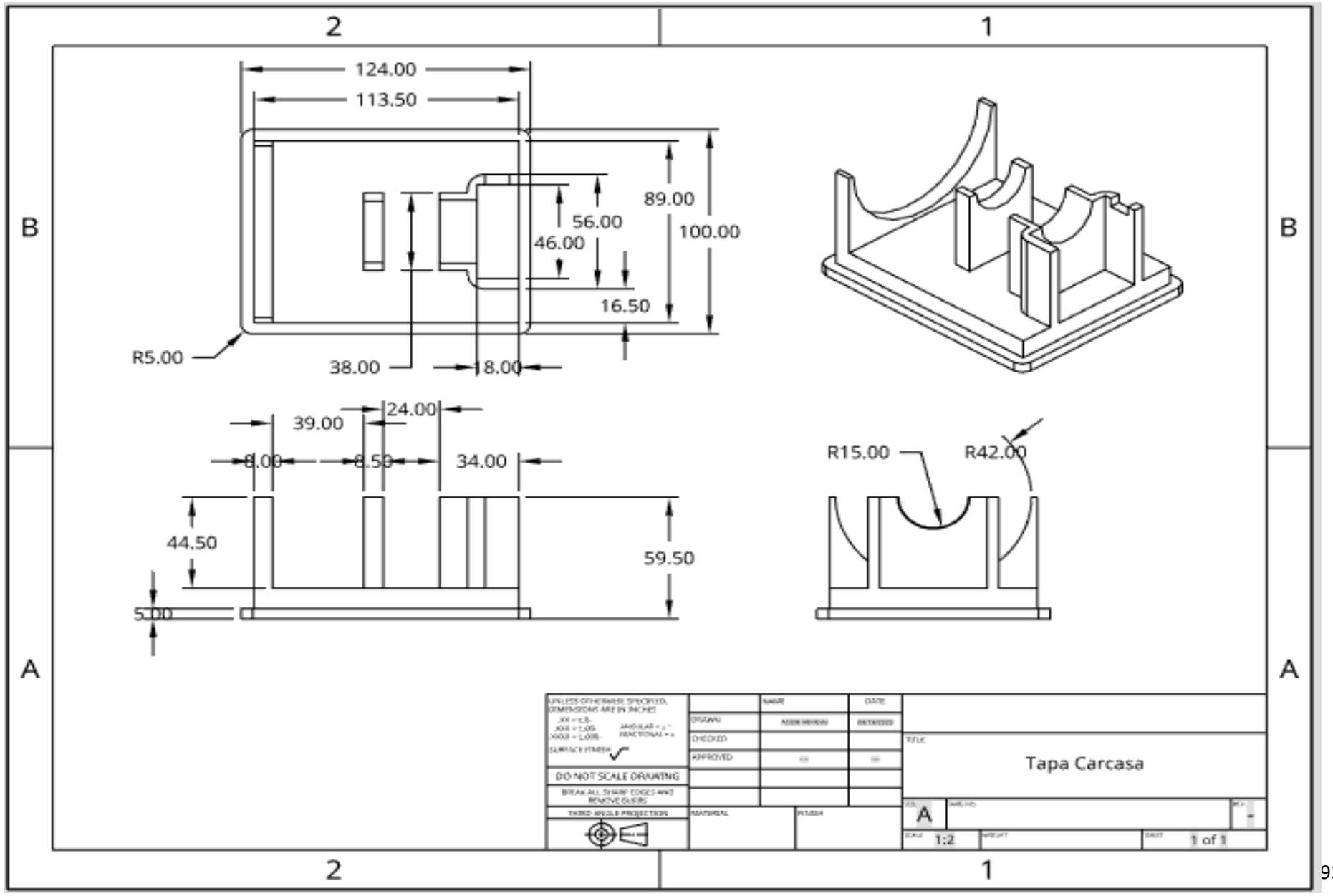
[28] Fan, S., Li, J., Zhang, Y., Tian, X., Wang, Q., He, X., Zhang, C., Huang, W., “On line detection of defective apples using computer vision system combined with deep learning methods”. Journal of Food Engineering. vol. 286, no. 110102, 2020.

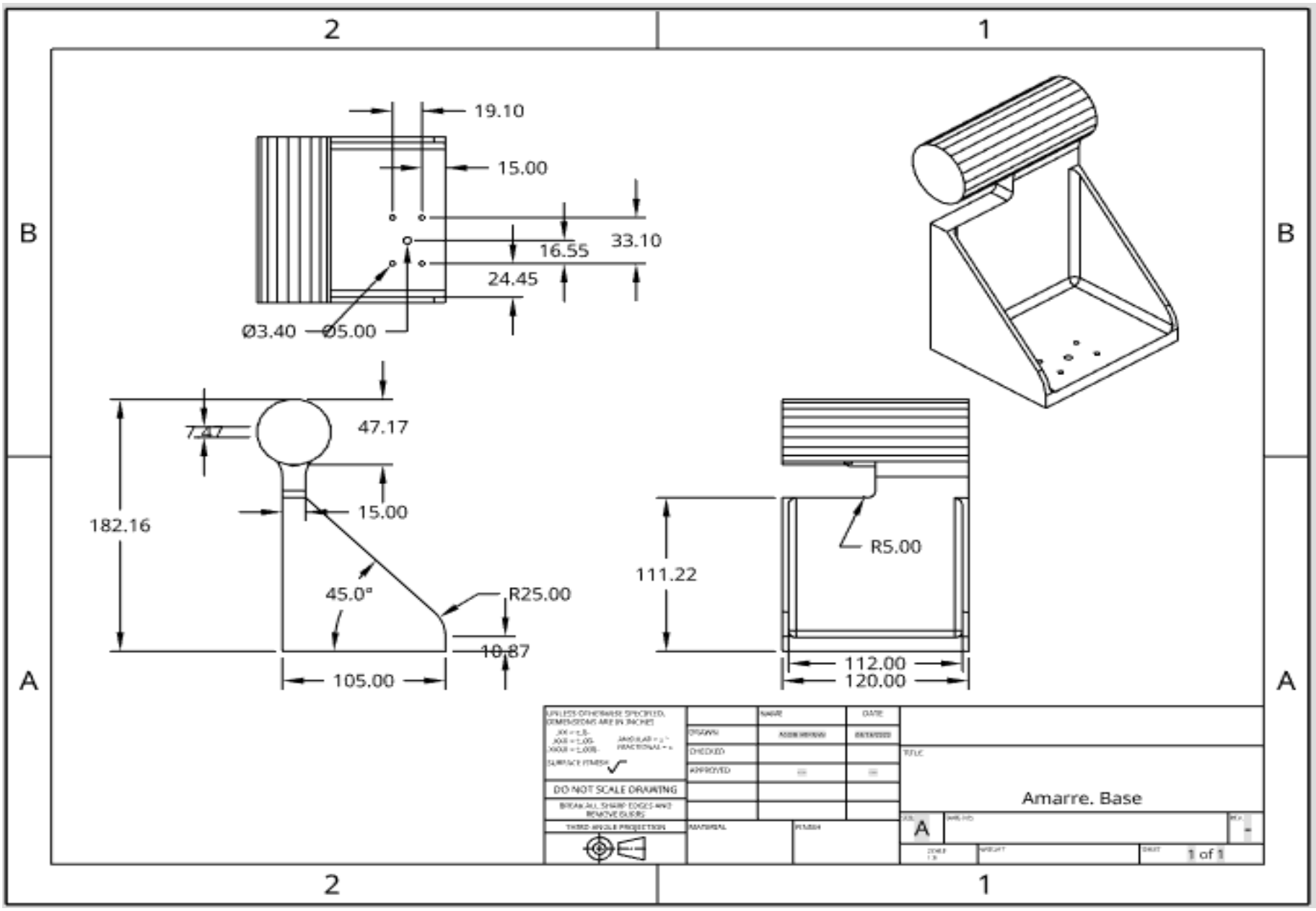
[29] Lee, S., Islam, N.U., Lee, S. “Robust image completion and masking with application to robotic bin picking”. Robotics and Autonomous System, vol 131, no. 103563, 2020.

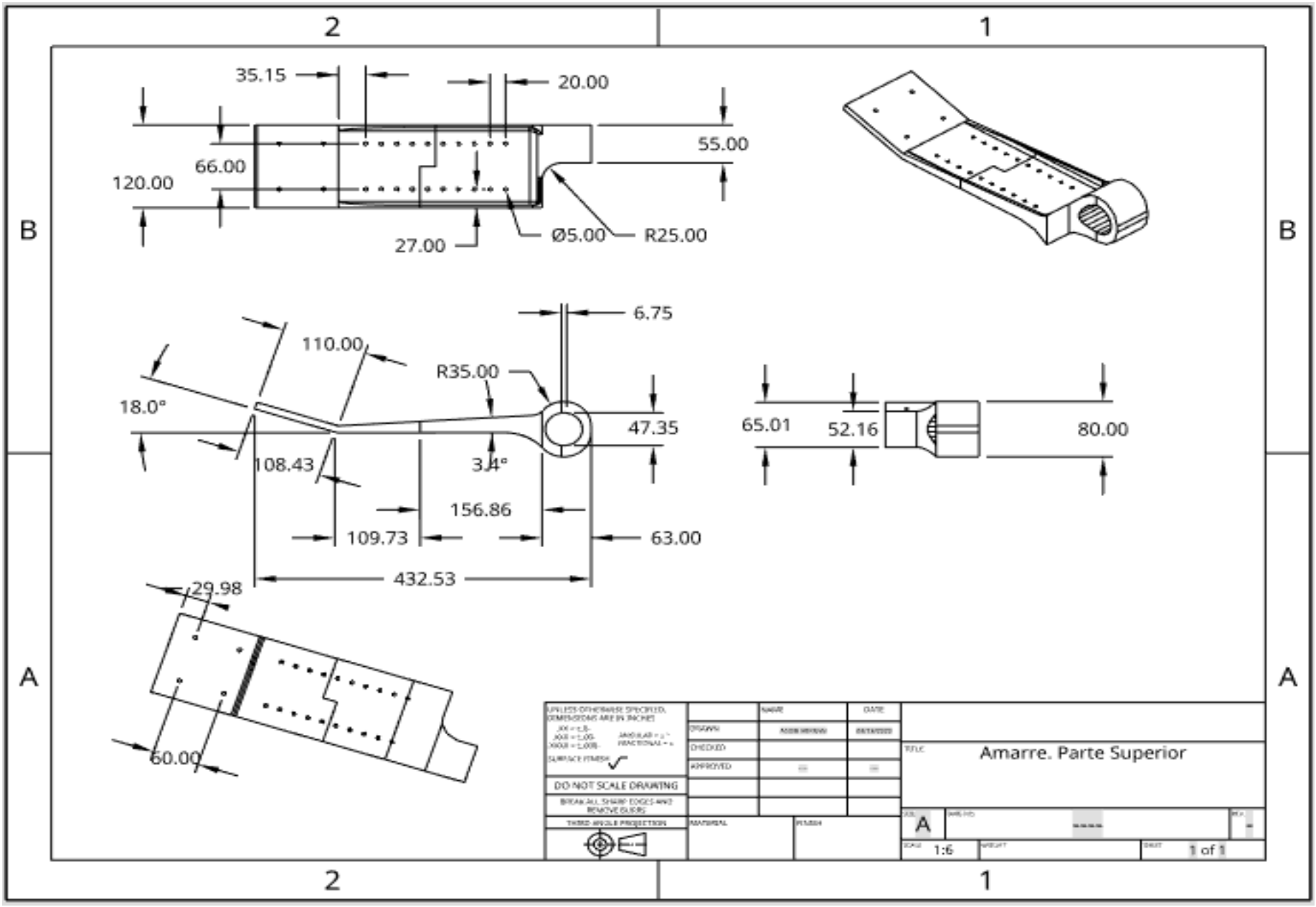
[30] Omron. Forpheus, 2020 .
https://www.omron.com/global/en/technology/information/forpheus/

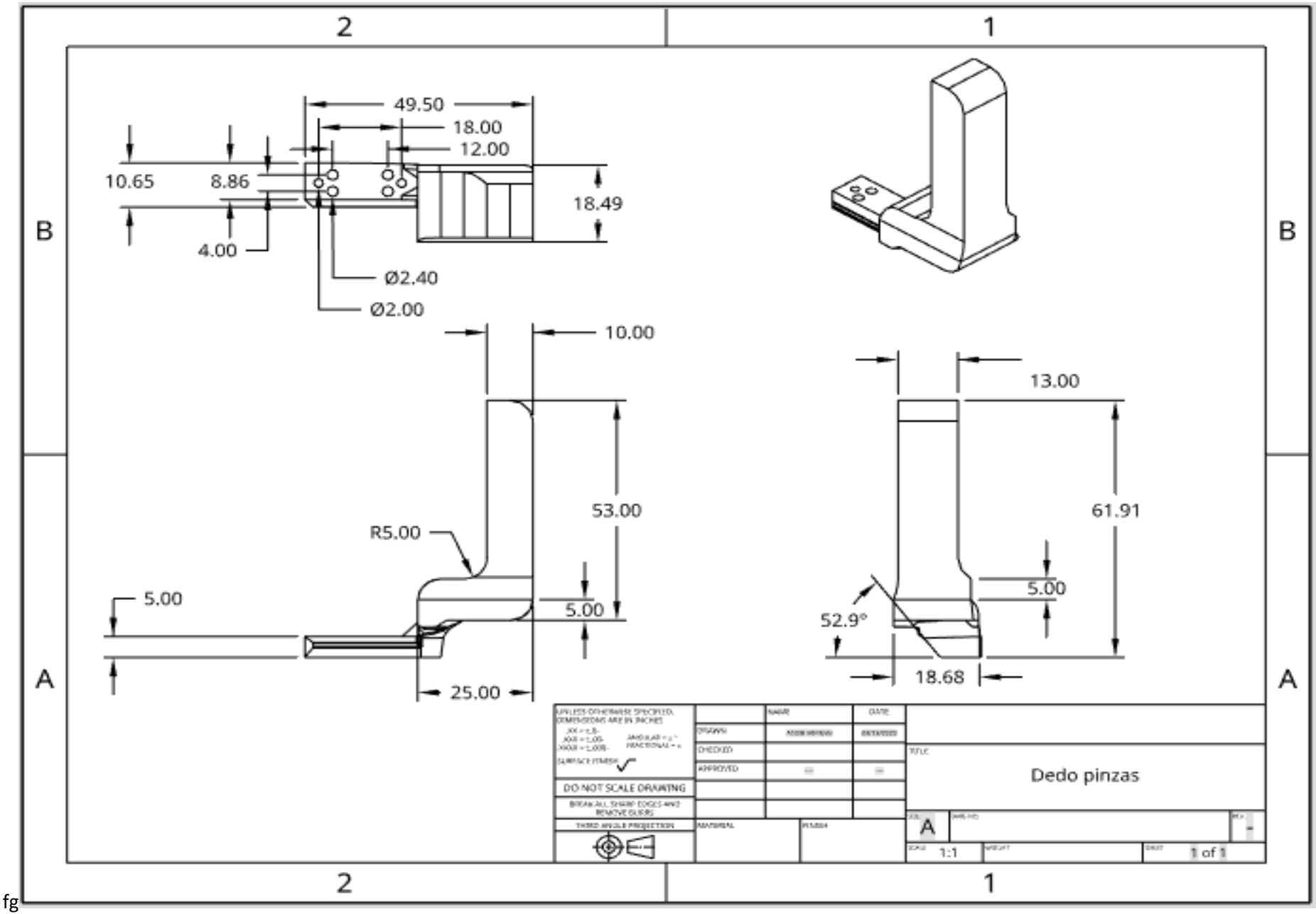
ANEXO I: PLANOS


2
1
B
A
R5.00
8.00
5.00
10.50
15.00
18.00
R7.50
38.00
58.00
23.50
6.70
R42.50
10.00
8.00
11.00
66.00
25.00
Ø5.00
R15.00
102.50
R25.50
R45.00
90.00
100.00
120.00
110.00
32.50
12.00
54.99
47.50
24.50
47.00
16.00
DO NOT SCALE DRAWING
Base Carcasa
A
1:2
1 of 1


2
1
B
A
124.00
113.50
89.00
100.00
56.00
46.00
16.50
R5.00
38.00
18.00
39.00
24.00
8.00
8.50
34.00
44.50
59.50
5.00
R15.00
R42.00
DO NOT SCALE DRAWING
THIRD ANGLE PROJECTION
DRAWN
CHECKED
APPROVED
NAME
DATE
Tapa Carcasa
A
1:2
1 of 1


2
1
B
A
19.10
15.00
33.10
16.55
24.45
Ø3.40
Ø5.00
47.17
7.47
182.16
15.00
45.0°
R25.00
10.87
105.00
111.22
R5.00
112.00
120.00
DO NOT SCALE DRAWING
THIRD ANGLE PROJECTION
Amarre. Base
A
1 of 1


2
1
B
A
35.15
20.00
55.00
66.00
120.00
27.00
Ø5.00
R25.00
6.75
110.00
R35.00
18.0°
47.35
108.43
3.4°
156.86
109.73
63.00
432.53
29.98
60.00
65.01
52.16
80.00
DO NOT SCALE DRAWING
THIRD ANGLE PROJECTION
Amarre. Parte Superior
A
1:6
1 of 1


2
1
B
A
49.50
18.00
12.00
10.65
8.86
18.49
4.00
Ø2.40
Ø2.00
10.00
53.00
R5.00
5.00
5.00
25.00
13.00
61.91
5.00
52.9°
18.68
DO NOT SCALE DRAWING
Dedo pinzas
A
1:1
1 of 1

fg

ANEXO II: CÓDIGO DE MATLAB Y ROBOT

A lo largo del anexo II se muestra el código utilizado para la realización del proyecto, tanto de Matlab como del Robot.

Código de Matlab

Programa principal

La estructura del código en Matlab consta de un programa principal sencillo del que se llama a las diferentes subrutinas. Tras el establecimiento de la conexión con el robot, se toma y procesa la primera imagen, obteniendo el tipo de cubo con el que se va a trabajar y su posición.

A partir de aquí, si el cubo es de 3x3 se ejecutará la primera parte de la sentencia IF que sonsiste en el envío de la posición al robot, la toma de imágenes, su procesamiento y su resolución y envío de los movimientos al robot.

Si el cubo a resolver es el de 2x2, la secuencia será análoga, cambiando únicamente las rutinas correspondientes al procesamiento de las imágenes y a la resolución del cubo.

```
%PROGRAMA FINAL
%Espera de conexión con Robot y toma de la primera foto
Toma_Imagen_Mesa;
%Procesamiento de la imagen.
%Obtención del tipo, la orientación y posición del cubo
Posicion_Cubo;

if Tipo_Cubo == 3
%Envío de la orienzación, posición y tipo del cubo al Robot
Envio_Posicion_Cubo;
%Toma de imágenes de las caras
Toma_Imagenes_Cubo;
%Procesamiento de las caras del cubo para la obtención de su estado
Procesamiento_Caras_Cubo;
%Resolución del cubo;
Algoritmo_Rubik_3x3;
%Envío de los movimientos al robot
Envio_Movimientos_Robot;
```

```
elseif Tipo_Cubo == 2
Posicion_Cubo;
%Envío de la orienzación, posición y tipo del cubo al Robot
Envio_Posicion_Cubo;
%Toma de imágenes de las caras
Toma_Imagenes_Cubo;
%Procesamiento de las caras del cubo para la obtención de su estado
Procesamiento_Caras_Cubo_2x2;
%Resolución del cubo;
Algoritmo_Rubik_2x2;
%Envío de los movimientos al robot
Envio_Movimientos_Robot;
end
```

Subprogramas

Toma_Imagenes_Mesa

```
tc=tcpip("192.168.125.1",14044);
fopen(tc);
senal=fread(tc);

vid = videoinput('winvideo', 1, 'RGB24_2560x1920');
vid.TriggerRepeat =1;
start(vid);

FotoMesa = getsnapshot(vid);
X=FotoMesa;
Y=imrotate(X,90);
Z=fliplr(Y);
stop(vid);
```

Posicion_Cubo

```
X=FotoMesa;
Y=imrotate(X,90);
Z=fliplr(Y);

Z2=imcrop(Z,[120 600 1900 1750]);

I = rgb2hsv(Z2);

% Define thresholds for channel 1 based on histogram settings
channel1Min = 0.000;
channel1Max = 1.000;

% Define thresholds for channel 2 based on histogram settings
channel2Min = 0.350;
channel2Max = 1.000;

% Define thresholds for channel 3 based on histogram settings
channel3Min = 0.000;
channel3Max = 1.000;

% Create mask based on chosen histogram thresholds
sliderBW = (I(:,:,1) >= channel1Min ) & (I(:,:,1) <= channel1Max) & ...
    (I(:,:,2) >= channel2Min ) & (I(:,:,2) <= channel2Max) & ...
    (I(:,:,3) >= channel3Min ) & (I(:,:,3) <= channel3Max);
BW = sliderBW;

BW = medfilt2(BW);
se=strel('square',20);
BW=imopen(BW,se);

K=cornermetric(BW,'MinimumEigenvalue');

corner_peaks=imregionalmax(K);

[L,Ne] = bwlabel(corner_peaks);
figure
imshow(corner_peaks)

prop=regionprops(L);
numItemMesa=0;
posItemMesa=zeros(27,2);
hold on
```

```
for n=1:length(prop);

    x=prop(n).Centroid(1);
    y=prop(n).Centroid(2);
    rectangle('Position',prop(n).BoundingBox,'EdgeColor','Green','LineWidth',2);
    plot(x,y,'*');
    numItemMesa=numItemMesa+1;
    posItemMesa(numItemMesa)=(x);
    posItemMesa(numItemMesa,2)=(y);

    posItemMesaX (n)= (x);
    posItemMesaY (n)= (y);

    posCuboX=0;
    posCuboY=0;

end

posItemMesaX2=0;
posItemMesaY2=0;
k=1;
j=1;
i=1;
    while  length(posItemMesaX2) <3
        if posItemMesaX(k)>0
            posItemMesaX2(j)=posItemMesaX(k);
             posItemMesaY2(i)=posItemMesaY(k);
            i=i+1;
            k=k+1;
            j=j+1;
        else
            k=k+1;
        end

    end

    posItemMesaX2;

    [posCuboY2,I]=max(posItemMesaY2);
    posCuboX2=posItemMesaX2(I);
    posCuboY2;
    posCuboX2;

    [posCuboY,I]=max(posItemMesaY);
    posCuboY;
    posCuboX=posItemMesaX(I);


hold off

angulo=atan((posCuboY-posCuboY2)/(posCuboX-posCuboX2))*(180/pi);

posCubo2=(posCuboY+posCuboY2)/2;
posCubo1=(posCuboX+posCuboX2)/2;

posCuboFinX= posCubo1 +(posCuboX-posCuboX2)/2/cosd(angulo)*cosd(90-angulo);
posCuboFinY= posCubo2 -(posCuboY-posCuboY2)/2/sind(angulo)*sind(90-angulo);
```

Envio_Posicion_Cubo

```
datosA=mat2str(PosCuboFinX);
datosB=mat2str(PosCuboFinY);
datosC=mat2str(angulo);
datosD=mat2str(Tipo_Cubo);

tc=tcpip("192.168.125.1",14043);

fopen(tc);

fwrite(tc,datosA);
pause(1);
fwrite(tc,datosB);
pause(1);
fwrite(tc,datosC);
pause(1);
fwrite(tc,datosD);

tclose(tp);
```

Toma_Imagenes_Caras

```
foto=50;

tc=tcpip("192.168.125.1",14044);
fopen(tc);

num_foto=fread(tc);
vid = videoinput('winvideo', 1, 'RGB24_2560x1920');
vid.TriggerRepeat =1;
start(vid);

Cara1 = getsnapshot(vid);
X=Cara1;
Y=imrotate(X,90);
Z=fliplr(Y);
imshow(Z)
stop(vid);

fwrite(tc,foto);

foto=foto+1
num_foto=fread(tc);
vid = videoinput('winvideo', 1, 'RGB24_2560x1920');
vid.TriggerRepeat =1;
start(vid);

Cara2 = getsnapshot(vid);
X=Cara2;
Y=imrotate(X,90);
Z=fliplr(Y);
imshow(Z)
stop(vid);

fwrite(tc,foto);

foto=foto+1
num_foto=fread(tc);
vid = videoinput('winvideo', 1, 'RGB24_2560x1920');
vid.TriggerRepeat =1;

start(vid);

Cara3 = getsnapshot(vid);
X=Cara3;
Y=imrotate(X,90);
Z=fliplr(Y);
imshow(Z)
stop(vid);
fwrite(tc,foto);
fclose(tc);
```

```
%Conexión con brazo izquierdo
tc=tcpip("192.168.125.1",14043);

fopen(tc);

foto=foto+1
num_foto=fread(tc);
vid = videoinput('winvideo', 1, 'RGB24_2560x1920');
vid.TriggerRepeat =1;

start(vid);

Cara4 = getsnapshot(vid);
X=Cara4;
Y=imrotate(X,90);
Z=fliplr(Y);
imshow(Z)
stop(vid);
fwrite(tc,foto);

foto=foto+1
num_foto=fread(tc);
vid = videoinput('winvideo', 1, 'RGB24_2560x1920');
vid.TriggerRepeat =1;
start(vid);

Cara5 = getsnapshot(vid);
X=Cara5;
Y=imrotate(X,90);
Z=fliplr(Y);
imshow(Z)
stop(vid);

fwrite(tc,foto);

foto=foto+1
num_foto=fread(tc);
vid = videoinput('winvideo', 1, 'RGB24_2560x1920');
vid.TriggerRepeat =1;

start(vid);

Cara6 = getsnapshot(vid);
X=Cara6;
Y=imrotate(X,90);
Z=fliplr(Y);
imshow(Z)
stop(vid);
```

Procesamiento_Caras_Cubo

```
CaraX=Cara1;
Proc_CaraF;
Cara_D=Cara;
Cubo(:,:,6)=Cara_D;

CaraX=Cara2;
Proc_CaraF;
Cara_L(1,1)=Cara(3,1);Cara_L(2,1)=Cara(3,2);Cara_L(3,1)=Cara(3,3);
Cara_L(1,2)=Cara(2,1);Cara_L(2,2)=Cara(2,2);Cara_L(3,2)=Cara(2,3);
Cara_L(1,3)=Cara(1,1);Cara_L(2,3)=Cara(1,2);Cara_L(3,3)=Cara(1,3);
Cubo(:,:,4)=Cara_L;

CaraX=Cara3;
Proc_CaraF;
Cara_U=Cara;
Cubo(:,:,5)=Cara_U;

CaraX=Cara4;
Proc_CaraF;
Cara_F=Cara;
Cubo(:,:,1)=Cara_F;

CaraX=Cara5;
Proc_CaraF;
Cara_R(1,1)=Cara(3,3);Cara_R(2,1)=Cara(2,3);Cara_R(3,1)=Cara(1,3);
Cara_R(1,2)=Cara(3,2);Cara_R(2,2)=Cara(2,2);Cara_R(3,2)=Cara(1,2);
Cara_R(1,3)=Cara(3,1);Cara_R(2,3)=Cara(2,1);Cara_R(3,3)=Cara(1,1);
Cubo(:,:,2)=Cara_R;

CaraX=Cara6;
Proc_CaraF;
Cara_B(1,1)=Cara(3,3);Cara_B(2,1)=Cara(2,3);Cara_B(3,1)=Cara(1,3);
Cara_B(1,2)=Cara(3,2);Cara_B(2,2)=Cara(2,2);Cara_B(3,2)=Cara(1,2);
Cara_B(1,3)=Cara(3,1);Cara_B(2,3)=Cara(2,1);Cara_B(3,3)=Cara(1,1);
Cubo(:,:,3)=Cara_B;
```

Proc_Cara_F

```
%PROCESAMIENTO CARA

Cara=[5 5 5;5 5 5;5 5 5];

Proc_Rojo_Caral;
Pos_Rojo_Cara;

Proc_Azul_Cara;
Pos_Azul_Cara;

Proc_Naranja_Cara;
Pos_Naranja_Cara;

Proc_Verde_Cara;
Pos_Verde_Cara;

Proc_Amarillo_Cara;
Pos_Amarillo_Cara;
```

Proc_Rojo_Cara1

```
X=CaraX;
Y=imrotate(X,90);
Z=fliplr(Y);
% Convert RGB image to chosen color space
I = rgb2hsv(Z);

% Define thresholds for channel 1 based on histogram settings
channel1Min = 0.972;
channel1Max = 0.019;

% Define thresholds for channel 2 based on histogram settings
channel2Min = 0.753;
channel2Max = 1.000;

% Define thresholds for channel 3 based on histogram settings
channel3Min = 0.000;
channel3Max = 1.000;

% Create mask based on chosen histogram thresholds
sliderBW = ( (I(:,:,1) >= channel1Min) | (I(:,:,1) <= channel1Max) ) & ...
    (I(:,:,2) >= channel2Min ) & (I(:,:,2) <= channel2Max) & ...
    (I(:,:,3) >= channel3Min ) & (I(:,:,3) <= channel3Max);
BW = sliderBW;

[L,Ne] = bwlabel(BW);

figure
imshow(label2rgb(BW ))

prop=regionprops(L);
numItemRed =0;
posItemRed =zeros(9,2);
posItemRedX=0;
posItemRedY=0;
hold on

for n=1:length(prop);
    x=prop(n).Centroid(1);
    y=prop(n).Centroid(2);
    if prop(n).Area> 5000
    rectangle('Position',prop(n).BoundingBox,'EdgeColor','Red','LineWidth',2);
    plot(x,y,'*');
    numItemRed =numItemRed+1;
    posItemRed (numItemRed)=(x);
    posItemRed (numItemRed,2)=(y);

    posItemRedX (numItemRed)= (x);
    posItemRedY (numItemRed)= (y);
    end
end

 title (['Hay ' num2str(numItemRed )  ' objetos rojos'])
 hold off
```

Pos_Rojo_Cara

```
if posItemRedX~=0

for i=1:length(posItemRedX)

       %CARA F
if  posItemRedX(i) <= 550
    if  posItemRedY(i) < 750
    Cara(1,1)=1
    end
    if posItemRedY(i) > 750 & posItemRedY(i) < 1000
    Cara(2,1)=1;
    end
    if posItemRedY(i) > 1000
    Cara(3,1)=1;
    end

end

if posItemRedX(i) > 550 & posItemRedX(i) < 800
    if  posItemRedY(i) < 750
    Cara(1,2)=1;
    end
    if posItemRedY(i) > 750 & posItemRedY(i) < 1000
    Cara(2,2)=1;
    end
    if posItemRedY(i) > 1000
    Cara(3,2)=1;
    end

end

if posItemRedX(i) >= 800
    if  posItemRedY(i) < 750
    Cara(1,3)=1;
    end
    if posItemRedY(i) > 750 & posItemRedY(i) < 1000
    Cara(2,3)=1;
    end
    if posItemRedY(i) > 1000
    Cara(3,3)=1;
    end


end

end
end
```

Algoritmo_Cubo_3x3

```
R=Cubo;

sol = Solve45_2(R);

solmat=cell2mat(sol);
movimientosR=zeros(length(sol));
movimientosR=diag(movimientosR);
movimientosR=movimientosR';
solmat(length(solmat)+1)=0;

for i=1:length(solmat)-1

if solmat(i)=='F' & solmat(i+1) ~= 'F' & solmat(i+1) ~= 'F2' & solmat(i+1) ~= 'R' & solmat(i+1) ~= 'R2' ...
   & solmat(i+1) ~= 'B'  & solmat(i+1) ~= 'B2' & solmat(i+1) ~= 'L'  & solmat(i+1) ~= 'L2' & ...
   solmat(i+1) ~= 'U' & solmat(i+1) ~= 'U2' & solmat(i+1) ~= 'D'  & solmat(i+1) ~= 'D2'
movimientosR(i)=11;
elseif solmat(i)=='F' & solmat(i+1)=='2'
movimientosR(i)=12;
elseif solmat(i)=='F'
 movimientosR(i)=10;

elseif solmat(i)=='R' & solmat(i+1) ~= 'F' & solmat(i+1) ~= 'F2' & solmat(i+1) ~= 'R'  & ...
       solmat(i+1) ~= 'R2' & solmat(i+1) ~= 'B'  & solmat(i+1) ~= 'B2' & solmat(i+1) ~= 'L'  & ...
       solmat(i+1) ~= 'L2' & solmat(i+1) ~= 'U'  & solmat(i+1) ~= 'U2' & solmat(i+1) ~= 'D'  & ...
       solmat(i+1) ~= 'D2'
movimientosR(i)=21;
elseif solmat(i)=='R'  & solmat(i+1)=='2'
movimientosR(i)=22;
elseif solmat(i)=='R'
movimientosR(i)=20;

elseif solmat(i)=='B' & solmat(i+1) ~= 'F'  & solmat(i+1) ~= 'F2' & solmat(i+1) ~= 'R'  & ...
       solmat(i+1) ~= 'R2' & solmat(i+1) ~= 'B'  & solmat(i+1) ~= 'B2' & solmat(i+1) ~= 'L'  & ...
       solmat(i+1) ~= 'L2' & solmat(i+1) ~= 'U'  & solmat(i+1) ~= 'U2' & solmat(i+1) ~= 'D'  & ...
       solmat(i+1) ~= 'D2'
movimientosR(i)=31;
elseif solmat(i)=='B' & solmat(i+1)=='2'
movimientosR(i)=32;
elseif solmat(i)=='B'
movimientosR(i)=30;

elseif solmat(i)=='L' & solmat(i+1) ~= 'F'  & solmat(i+1) ~= 'F2' & solmat(i+1) ~= 'R'  & ...
       solmat(i+1) ~= 'R2' & solmat(i+1) ~= 'B'  & solmat(i+1) ~= 'B2' & solmat(i+1) ~= 'L'  & ...
       solmat(i+1) ~= 'L2' & solmat(i+1) ~= 'U'  & solmat(i+1) ~= 'U2' & solmat(i+1) ~= 'D'  & ...
       solmat(i+1) ~= 'D2'
movimientosR(i)=41;
elseif solmat(i)=='L'  & solmat(i+1)=='2'
movimientosR(i)=42;
elseif solmat(i)=='L'
movimientosR(i)=40;

elseif solmat(i)=='U' & solmat(i+1) ~= 'F'  & solmat(i+1) ~= 'F2' & solmat(i+1) ~= 'R'  & ...
       solmat(i+1) ~= 'R2' & solmat(i+1) ~= 'B'  & solmat(i+1) ~= 'B2' & solmat(i+1) ~= 'L'  & ...
       solmat(i+1) ~= 'L2' & solmat(i+1) ~= 'U'  & solmat(i+1) ~= 'U2' & solmat(i+1) ~= 'D'  & ...
       solmat(i+1) ~= 'D2'
movimientosR(i)=51;
elseif solmat(i)=='U' & solmat(i+1)=='2'
movimientosR(i)=52;
elseif solmat(i)=='U'
movimientosR(i)=50;

elseif solmat(i)=='D' & solmat(i+1) ~= 'F'  & solmat(i+1) ~= 'F2' & solmat(i+1) ~= 'R'  & ...
       solmat(i+1) ~= 'R2' & solmat(i+1) ~= 'B'  & solmat(i+1) ~= 'B2' & solmat(i+1) ~= 'L'  & ...
       solmat(i+1) ~= 'L2' & solmat(i+1) ~= 'U'  & solmat(i+1) ~= 'U2' & solmat(i+1) ~= 'D'  & ...
       solmat(i+1) ~= 'D2'
movimientosR(i)=61;
elseif solmat(i)=='D' & solmat(i+1)=='2'
movimientosR(i)=62;
elseif solmat(i)=='D'
movimientosR(i)=60;

end
end
```

```
noblancos=0;
for j=1:length(movimientosR)
if  movimientosR(j)~=0
noblancos=noblancos+1;
end
end

movimientosR1=zeros(noblancos);
movimientosR1=diag(movimientosR1);
k=1;
for j=1:length(movimientosR)
if  movimientosR(j)~=0
movimientosR1(k)=movimientosR(j);
k=k+1;
end
end

movimientosR1=movimientosR1';
```

Solve_45

```
function sol = Solve45_2(R)
% Solve the cube using T45

P = load('Prunes');
P1 = P.P1;
P2 = P.P2;
P3 = P.P3;
P4 = P.P4;

sol = [];

E = GetEdges(R);
C = GetCorners(R);

%PHASE 1: CURE EDGES
moves = {'L'  ,'R'  ,'F'  ,'B'  ,'U'  ,'D';...
         'L2' ,'R2' ,'F2' ,'B2' ,'U2' ,'D2';...
         'L''','R''','F''','B''','U''','D'''};

n = State2Ind(E(2,:))+1;
N = P1(n);

while N>0
    for i=1:18
        E2 = TwistEdges(E,moves{i});
        n = State2Ind(E2(2,:))+1;
        M = P1(n);
        if M<N
            N = M;
            E = E2;
            C = TwistCorners(C,moves{i});
            sol = [sol moves(i)];
            break
        end
    end
end

%PHASE 2: MOVE LR-SLICE EDGES TO UD-SLICE + ORIENT CORNERS
moves = {'L'  ,'R'  ,'F'  ,'B'  ,'U2';...
         'L2' ,'R2' ,'F2' ,'B2' ,'D2';...
         'L''','R''','F''','B''','00'};

Clist = P.ClistP2;
Elist = P.ElistP2;

F = E(1,:);
```

```
F = double(F>=9);
Cind = State2Ind(C(2,:));
Eind = State2Ind(F);

n = Clist==Cind;
m = Elist==Eind;

N = P2(n,m);

while N>0
    for i=1:14
        C2 = TwistCorners(C,moves{i});
        E2 = TwistEdges(E,moves{i});
        F2 = E2(1,:);
        F2 = double(F2>=9);
        Cind = State2Ind(C2(2,:));
        Eind = State2Ind(F2(1,:));
        n = Clist==Cind;
        m = Elist==Eind;
        M = P2(n,m);
        if M<N
            N = M;
            C = C2;
            E = E2;
            sol = [sol moves(i)];
            break
        end
    end
end

%PHASE 3: FIX EDGES IN THEIR SLICE W/ EVEN PERMUTATION + FIX CORNERS IN
%         ORBIT W/ EVEN PERMUTATION
moves = {'L' ,'L'','L2',...
         'R' ,'R'','R2',...
         'F2','B2','U2','D2'};

Clist = P.ClistP3;
Elist = P.ElistP3;

F = ceil(E(1,1:8)/4)-1;

Cind = State2Ind(C(1,:));
Eind = State2Ind(F,2);

n = find(Clist==Cind);
m = find(Elist==Eind);
```

```
N = P3(n,m);

while N>0
    for i=1:10
        C2 = TwistCorners(C,moves{i});
        E2 = TwistEdges(E,moves{i});
        F = ceil(E2(1,1:8)/4)-1;
        Cind = State2Ind(C2(1,:));
        Eind = State2Ind(F,2);
        n = Clist==Cind;
        m = Elist==Eind;
        M = P3(n,m);
        if M<N
            N = M;
            C = C2;
            E = E2;
            sol = [sol moves(i)];
            break
        end
    end
end
```

```
%PHASE 4: SOLVE THE CUBE
moves = {'L2','R2','F2','B2','U2','D2'};

Clist = P.ClistP4;
Elist = P.ElistP4;

Cind = State2Ind(C(1,:));
Eind = State2Ind(E(1,:));

n = Clist==Cind;
m = Elist==Eind;
N = P4(n,m);

while N>0
    for i=1:6
        C2 = TwistCorners(C,moves{i});
        E2 = TwistEdges(E,moves{i});
        Cind = State2Ind(C2(1,:));
        Eind = State2Ind(E2(1,:));
        n = Clist==Cind;
        m = Elist==Eind;
        M = P4(n,m);
        if M<N
            N = M;
            C = C2;
            E = E2;
            sol = [sol moves(i)];
            break
        end
    end
end

sol = rubopt(sol);
```

Envio_Movimientos_Robot

```
num_mov=length(movimientosR1);
datos=mat2str(num_mov);

if length(movimientosR1) > 26
    j=1;
    for i=27:length(movimientosR1)
        movimientosR2(j)=movimientosR1(i);
        j=j+1;
    end
    for i=length(movimientosR1):-1:27
        movimientosR1(i)=[];
    end
end

if num_mov < 27
datos1=mat2str(movimientosR1);

tc=tcpip("192.168.125.1",14044);

fopen(tc);

fwrite(tc,datos);
pause(1);
fwrite(tc,datos1);

else
datos1=mat2str(movimientosR1);
datos2=mat2str(movimientosR2);

tc=tcpip("192.168.125.1",14044);

fopen(tc);

fwrite(tc,datos);
pause(1);
fwrite(tc,datos1);
pause(1);
fwrite(tc,datos2);
end

pause(3);
%Brazo Izquierdo

if num_mov < 27

datos1=mat2str(movimientosR1);

tc=tcpip("192.168.125.1",14043);
```

```
fopen(tc);

fwrite(tc,datos);
pause(1);
fwrite(tc,datos1);

else
datos1=mat2str(movimientosR1);
datos2=mat2str(movimientosR2);

tc=tcpip("192.168.125.1",14043);

fopen(tc);

fwrite(tc,datos);
pause(1);
fwrite(tc,datos1);
pause(1);
fwrite(tc,datos2);
end

fclose(tc);
```

En caso de que el cubo a resolver sea el de 2x2, su algoritmo es el siguiente:

Solve222:

```
function [rot sol] = Solve222(R)
load('God222.mat')
rot={'y0','z0'; 'y1','z0';...
     'y0','z1'; 'y1','z1';...
     'y0','z2'; 'y1','z2';...
     'y0','z3'; 'y1','z3';...
     'y2','z0'; 'y3','z0';...
     'y2','z1'; 'y3','z1';...
     'y2','z2'; 'y3','z2';...
     'y2','z3'; 'y3','z3';...
     'x1','z0'; 'x3','z0';...
     'x1','z1'; 'x3','z1';...
     'x1','z2'; 'x3','z2';...
     'x1','z3'; 'x3','z3'};
move = {'R'  ,'F'  ,'D';...
        'R2' ,'F2' ,'D2';...
        'R''','F''','D'''};
C = GetCorners2(R);
t = 1;
while C(1,1)~=1 || C(2,1)~=0
    t = t+1;
    R1 = rubrot2(R,rot(t,:));
    C = GetCorners2(R1);
end
```

```
rot = rot(t,:);
i1 = State2Ind(C(1,:));
i2 = State2Ind(C(2,:));
n = find(list1==i1);
m = find(list2==i2);
N = A(n,m);
sol = [];
while N>0
    for i=1:9
        C1 = TwistCorners(C,move{i});
        i1 = State2Ind(C1(1,:));
        i2 = State2Ind(C1(2,:));
        n = find(list1==i1);
        m = find(list2==i2);
        M = A(n,m);
        if M<N
            N = M;
            C = C1;
            sol = [sol move(i)];
            break
        end
    end
end
% disp(sol)
```

Código del robot Yumi

Brazo izquierdo

Debido a que en la célula de montaje coexisten tres proyectos, el programa principal consiste en el código que determina el programa a ejecutar en función de las posiciones que el usuario elija en los *switches.*

Una vez seleccionado nuestro programa, se realiza la llamada al programa principal.

Main Module

```
MODULE MainModule

    !DECLARACIÓN DE VARIABLES
    !Indicador de aplicación
    PERS num Programa:=0;
    !Variables de sincronización de inicio
    PERS tasks tareas{2}:=[["T_ROB_L"],["T_ROB_R"]];
    VAR syncident s_inicio;
    VAR syncident eslabones;
    VAR syncident extraccion;
    !Trabajo de visión artificial para detectar los pernos
    CONST string trabajo:="Pernos_bueno.job";
    !FIN DECLARACIÓN DE VARIABLES

    PROC main()

        WHILE TRUE DO

            IF custom_DI_0=0 THEN
                !Si el selector de 2P lo indica, se inicia esta aplicación
                !Se cambia el indicador de programa
                Programa:=0;
                !Reproducción mensaje de bienvenida
                sound_iniciar:=TRUE;
                !Se carga el trabajo de visión en la camara
                CamSetProgramMode HandCameraL;
                CamLoadJob HandCameraL,trabajo;
                CamSetRunMode HandCameraL;
                !Se espera a que se pulse el pulsador verde para iniciar el proceso
                WaitDI custom_DI_3,1;
                !Se enciende el indicador verde
                SetDO custom_DO_0,1;
                !Se apaga el indicador rojo
                SetDO custom_DO_1,0;
                !Se espera al brazo derecho para iniciar a la vez
                WaitSyncTask s_inicio,tareas;
                !Procedimiento inicial (solo la primera vez)
                inicio;
                WHILE custom_DI_0=0 DO
                    !Mientas el selector de 2P lo indique: bucle repetitivo
                    !Se lee el selector de 3P para saber el numero de eslabones a ensamblar,
                    !indicando el numero de eslabones a ensamblar por el sistema de avisos por voz
                    IF custom_DI_2=1 THEN
                        sound_dos_piezas:=TRUE;
                        no_eslabones:=2;
                    ELSEIF custom_DI_1=1 THEN
                        sound_cuatro_piezas:=TRUE;
                        no_eslabones:=4;
                    ELSE
                        sound_tres_piezas:=TRUE;
                        no_eslabones:=3;
                    ENDIF
                    !El brazo derecho espera a que se sepa el numero de eslabones a ensamblar
```

```
                WaitSyncTask eslabones,tareas;

                FOR i FROM 1 TO no_eslabones DO
                    !Se intercambia el eslabón con el brazo derecho
                    coger_eslabon;
                    !Se situa el eslabón en el molde de ensamblaje
                    situar_eslabon(i);
                ENDFOR

                FOR i FROM 1 TO (no_eslabones-1) DO
                    !Se localizan los pernos
                    foto;
                    !Se extraen los pernos del almacén
                    cojer_perno;
                    !Se ensamblan los pernos
                    ensamblar_pernos(i);
                ENDFOR
                !Se espera a que los dos brazos esten listos para la extracción de la cadena ensamblada
                WaitSyncTask extraccion,tareas;
                !Se extrae la cadena ensamblada y se deja en la cinta
                dejar_cinta;
             ENDWHILE

        ELSEIF custom_DI_0=1 AND custom_DI_2=1 THEN
            !ON
            Programa:=1;
            !!-----------------------------------------------------------------------------------------------------------------------
            !!OIHANE

            Inicio_L;

            WHILE custom_DI_0=1 DO

                CamaraEnUso:=TRUE;
                ! Pulsar boton verde cuando esten todas las piezas en su sitio
                IF custom_DI_3=1 THEN

                    CocheAmarillo_L;
                    CocheRojo_L;
                    CocheVerde_L;
                    sincroHome_L;

                ENDIF

            ENDWHILE

             !!-----------------------------------------------------------------------------------------------------------------------

             ELSEIF custom_DI_0=1 AND custom_DI_2=1 THEN
             !ON
             Programa:=2;
             !!-----------------------------------------------------------------------------------------------------------------------
             !!Asier
             !!Se ejecuta el programa de resolución de cubos de rubik
             Principal;

        ENDIF

    ENDWHILE

ENDPROC

ENDMODULE
```

Programa Principal

El programa principal está formado por tres subrutinas. En primer lugar, la encargada de recibir el cubo por parte del brazo derecho y de tomar las fotos a sus caras. Hecho esto, se comunicará con Matlab para recibir en array con los movimientos a realizar, y finalmente, procesará dicho array y junto con el brazo derecho ejecutará los movimientos que resuelvan el cubo.

```
PROC Principal ()
    Toma_Fotos_L;              !Recepción del cubo y toma de las fotos
    Recibir_Movimientos_L;     !Conexión con Matlab y recepción de los movimientos
    Resolución_Cubo_L;         !Sincronización con brazo R y ejecución de los movimientos
ENDPROC
```

Subprogramas

Toma_Fotos_L

```
WaitTime 3;
WaitSyncTask sync_RL_A, ListaTareas;
    !Crear comunicacion
    SocketCreate server1;
    SocketBind server1, "192.168.125.1",14043;
    SocketListen server1;
    SocketAccept server1,client1;

Foto1;
!enviar mensaje al cliente
SocketSend client1,\str:="D";
!recivir mensaje
SocketReceive client1,\RawData:=data;
UnpackRawBytes data,1,fotos\Ascii:=2;
oki:=StrtoVal(fotos,numfoto);
WaitUntil numfoto=5;

Foto2;
!enviar mensaje al cliente
SocketSend client1,\str:="E";
!recivir mensaje
SocketReceive client1,\RawData:=data;
UnpackRawBytes data,1,fotos\Ascii:=2;
oki:=StrtoVal(fotos,numfoto);
WaitUntil numfoto=6;

 Foto3;
 !enviar mensaje al cliente
 SocketSend client1,\str:="F";
 !recivir mensaje
 SocketReceive client1,\RawData:=data;
 UnpackRawBytes data,1,fotos\Ascii:=2;
 oki:=StrtoVal(fotos,numfoto);
 WaitUntil numfoto=7;

 !Cerrar comunicacion
 SocketClose server1;

MoveL Agarre_L_1,v300,z100,Servo_L\WObj:=WO_Cubo;
MoveL Agarre_L2_1,v300,z100,Servo_L\WObj:=WO_Cubo;
WaitTime 3;
WaitSyncTask sync_RL_B, ListaTareas;
Agarre_L6;
WaitSyncTask sync_RL_C, ListaTareas;
WaitTime 3;
```

Recibir_Movimientos_L

```
        SocketCreate server1; !Creación del socket vinculado al servidor Server1
        SocketBind server1, "192.168.125.1",14044; !Definición del puerto y la IP del socket
        SocketListen server1; !Espera a la confirmación del cliente
        SocketAccept server1,client1; !Creación de la comunicación

   SocketReceive client1,\RawData:=data;
   UnpackRawBytes data,1,Recmov\Ascii:=2;
   oki:=StrToVal(Recmov,nummov);

   IF nummov<27 THEN
   mov:=nummov*3+1;
   SocketReceive client1,\RawData:=datos;
   UnpackRawBytes datos,1,Recmov1\Ascii:=mov;

    FOR j FROM 1 TO nummov DO
     part:=StrPart(Recmov1,B,2);
     ok:=StrToVal(part,vec{j});
     B:=B+3;
    ENDFOR
  ELSE
      SocketReceive client1,\RawData:=datos;
      UnpackRawBytes datos,1,Recmov1\Ascii:=79;

   FOR j FROM 1 TO 26 DO
    part:=StrPart(Recmov1,B,2);
    ok:=StrToVal(part,vec{j});
    B:=B+3;
   ENDFOR

   nummov:=nummov-26;
   mov:=nummov*3+1;

   SocketReceive client1,\RawData:=datos;
  UnpackRawBytes datos,1,Recmov1\Ascii:=mov;

  B:=2;
   FOR j FROM 1 TO nummov DO
    part:=StrPart(Recmov1,B,2);
    ok:=StrToVal(part,vec1{j});
    B:=B+3;
   ENDFOR
  ENDIF

       SocketClose server1;

           WaitSyncTask sync_RL_8, ListaTareas_RL;

    ENDPROC


ENDMODULE
```

Resolución_Cubo_L

```
FOR i FROM 1 TO nummov DO
 IF CuboTres=0 THEN

   IF vec{i}=40 THEN
    WaitTime 3;
    Girar_L;
    WaitSyncTask sync_RL_E, ListaTareas;

   ELSEIF vec{i}=41 THEN
    WaitTime 3;
    Girar_L1;
    WaitSyncTask sync_RL_E, ListaTareas;

   ELSEIF vec{i}=42 THEN
    WaitTime 3;
    Girar_L2;
    WaitSyncTask sync_RL_E, ListaTareas;

   ELSEIF vec{i}=50 THEN
    WaitTime 3;
    Girar_U;
    WaitSyncTask sync_RL_E, ListaTareas;
```

```
ELSEIF vec{i}=51 THEN
 WaitTime 3;
 Girar_U1;
 WaitSyncTask sync_RL_E, ListaTareas;

ELSEIF vec{i}=52 THEN
 WaitTime 3;
 Girar_U2;
 WaitSyncTask sync_RL_E, ListaTareas;

ELSEIF vec{i}=60 THEN
 MoveL Apro_L_1, v300,fine,Servo_L;
 WaitSyncTask sync_RL_E, ListaTareas;
 Girar_D;
 WaitSyncTask sync_RL_F, ListaTareas;
 WaitSyncTask sync_RL_G, ListaTareas;

ELSEIF vec{i}=61 THEN
 MoveL Apro_L_1, v300,fine,Servo_L;
 WaitSyncTask sync_RL_E, ListaTareas;
 Girar_D1;
 WaitSyncTask sync_RL_F, ListaTareas;
 WaitSyncTask sync_RL_G, ListaTareas;

ELSEIF vec{i}=62 THEN
 MoveL Apro_L_1, v300,fine,Servo_L;
 WaitSyncTask sync_RL_E, ListaTareas;
 Girar_D2;
 WaitSyncTask sync_RL_F, ListaTareas;
 WaitSyncTask sync_RL_G, ListaTareas;
```

```
ELSEIF vec{i}=10 THEN
 WaitTime 3;
 Rotar_L;
 WaitSyncTask sync_RL_E, ListaTareas;
 WaitSyncTask sync_RL_F, ListaTareas;
 WaitTime 3;
 Rotar_L1;
 WaitSyncTask sync_RL_G, ListaTareas;

 ELSEIF vec{i}=11 THEN
 WaitTime 3;
 Rotar_L;
 WaitSyncTask sync_RL_E, ListaTareas;
 WaitSyncTask sync_RL_F, ListaTareas;
 WaitTime 3;
 Rotar_L1;
 WaitSyncTask sync_RL_G, ListaTareas;

 ELSEIF vec{i}=12 THEN
 WaitTime 3;
 Rotar_L;
 WaitSyncTask sync_RL_E, ListaTareas;
 WaitSyncTask sync_RL_F, ListaTareas;
 WaitTime 3;
 Rotar_L1;
 WaitSyncTask sync_RL_G, ListaTareas;

 ELSEIF vec{i}=20 THEN
 Agarre_L5;
 WaitSyncTask sync_RL_E,ListaTareas;

 ELSEIF vec{i}=21 THEN
 Agarre_L5;
 WaitSyncTask sync_RL_E,ListaTareas;

 ELSEIF vec{i}=22 THEN
 Agarre_L5;
 WaitSyncTask sync_RL_E,ListaTareas;

 ELSEIF vec{i}=30 THEN
 Agarre_L5;
 WaitSyncTask sync_RL_E,ListaTareas;

 ELSEIF vec{i}=31 THEN
 Agarre_L5;
 WaitSyncTask sync_RL_E,ListaTareas;

 ELSEIF vec{i}=32 THEN
 Agarre_L5;
 WaitSyncTask sync_RL_E,ListaTareas;


   ENDIF
  ENDIF
```

```
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!

  IF CuboTres=1 THEN

    IF vec{i}=40 THEN
     WaitTime 3;
     Girar_L_2;
     WaitSyncTask sync_RL_E, ListaTareas;

    ELSEIF vec{i}=41 THEN
     WaitTime 3;
     Girar_L1_2;
     WaitSyncTask sync_RL_E, ListaTareas;

    ELSEIF vec{i}=42 THEN
     WaitTime 3;
     Girar_L2_2;
     WaitSyncTask sync_RL_E, ListaTareas;

  ELSEIF vec{i}=50 THEN
   WaitTime 3;
   Girar_U_2;
   WaitSyncTask sync_RL_E, ListaTareas;

  ELSEIF vec{i}=51 THEN
   WaitTime 3;
   Girar_U1_2;
   WaitSyncTask sync_RL_E, ListaTareas;

  ELSEIF vec{i}=52 THEN
   WaitTime 3;
   Girar_U2_2;
   WaitSyncTask sync_RL_E, ListaTareas;


ELSEIF vec{i}=60 THEN

 IF vec{i-1}=40 OR vec{i-1}=41 OR vec{i-1}=42 OR vec{i-1}=50 OR vec{i-1}=51 OR vec{i-1}=52 THEN
 WaitTime 3;
 Rotar_L_2;
 WaitSyncTask sync_RL_E, ListaTareas;
 WaitSyncTask sync_RL_F, ListaTareas;
 WaitTime 3;
 Rotar_L1_2;
 WaitSyncTask sync_RL_G, ListaTareas;

 ELSE
 MoveL Apro_L_1_2, v300,fine,Servo_L;
 WaitSyncTask sync_RL_E, ListaTareas;
 Girar_D_2;
 WaitSyncTask sync_RL_F, ListaTareas;
 WaitSyncTask sync_RL_G, ListaTareas;
ENDIF
```

```
ELSEIF vec{i}=61 THEN

 IF vec{i-1}=40 OR vec{i-1}=41 OR vec{i-1}=42 OR vec{i-1}=50 OR vec{i-1}=51 OR vec{i-1}=52 THEN
 WaitTime 3;
 Rotar_L_2;
 WaitSyncTask sync_RL_E, ListaTareas;
 WaitSyncTask sync_RL_F, ListaTareas;
 WaitTime 3;
 Rotar_L1_2;
 WaitSyncTask sync_RL_G, ListaTareas;

 ELSE
 MoveL Apro_L_1_2, v300,fine,Servo_L;
 WaitSyncTask sync_RL_E, ListaTareas;
 Girar_D1_2;
 WaitSyncTask sync_RL_F, ListaTareas;
 WaitSyncTask sync_RL_G, ListaTareas;
ENDIF

ELSEIF vec{i}=62 THEN

 IF vec{i-1}=40 OR vec{i-1}=41 OR vec{i-1}=42 OR vec{i-1}=50 OR vec{i-1}=51 OR vec{i-1}=52 THEN
 WaitTime 3;
 Rotar_L_2;
 WaitSyncTask sync_RL_E, ListaTareas;
 WaitSyncTask sync_RL_F, ListaTareas;
 WaitTime 3;
 Rotar_L1_2;
 WaitSyncTask sync_RL_G, ListaTareas;

 ELSE
 MoveL Apro_L_1_2, v300,fine,Servo_L;
 WaitSyncTask sync_RL_E, ListaTareas;
 Girar_D2_2;
 WaitSyncTask sync_RL_F, ListaTareas;
 WaitSyncTask sync_RL_G, ListaTareas;
ENDIF
```

```
ELSEIF vec{i}=10 THEN
 WaitTime 3;
 Rotar_L_2;
 WaitSyncTask sync_RL_E, ListaTareas;
 WaitSyncTask sync_RL_F, ListaTareas;
 WaitTime 3;
 Rotar_L1_2;
 WaitSyncTask sync_RL_G, ListaTareas;

 ELSEIF vec{i}=11 THEN
 WaitTime 3;
 Rotar_L_2;
 WaitSyncTask sync_RL_E, ListaTareas;
 WaitSyncTask sync_RL_F, ListaTareas;
 WaitTime 3;
 Rotar_L1_2;
 WaitSyncTask sync_RL_G, ListaTareas;

 ELSEIF vec{i}=12 THEN
 WaitTime 3;
 Rotar_L_2;
 WaitSyncTask sync_RL_E, ListaTareas;
 WaitSyncTask sync_RL_F, ListaTareas;
 WaitTime 3;
 Rotar_L1_2;
 WaitSyncTask sync_RL_G, ListaTareas;
```

```
 ELSEIF vec{i}=20 THEN
 Agarre_L5_2;
 WaitSyncTask sync_RL_E,ListaTareas;

 ELSEIF vec{i}=21 THEN
 Agarre_L5_2;
 WaitSyncTask sync_RL_E,ListaTareas;

 ELSEIF vec{i}=22 THEN
 Agarre_L5_2;
 WaitSyncTask sync_RL_E,ListaTareas;

ELSEIF vec{i}=30 THEN

IF vec{i-1}=10 OR vec{i-1}=11 OR vec{i-1}=12 OR vec{i-1}=20 OR vec{i-1}=21 OR vec{i-1}=22 THEN
MoveL Apro_L_1_2, v300,fine,Servo_L;
WaitSyncTask sync_RL_E, ListaTareas;
Girar_B_2;
WaitSyncTask sync_RL_F, ListaTareas;
WaitSyncTask sync_RL_G, ListaTareas;

ELSE
Agarre_L5_2;
WaitSyncTask sync_RL_E,ListaTareas;
ENDIF

 ELSEIF vec{i}=31 THEN

 IF vec{i-1}=10 OR vec{i-1}=11 OR vec{i-1}=12 OR vec{i-1}=20 OR vec{i-1}=21 OR vec{i-1}=22 THEN
 MoveL Apro_L_1_2, v300,fine,Servo_L;
 WaitSyncTask sync_RL_E, ListaTareas;
 Girar_B1_2;
 WaitSyncTask sync_RL_F, ListaTareas;
 WaitSyncTask sync_RL_G, ListaTareas;

ELSE
Agarre_L5_2;
WaitSyncTask sync_RL_E,ListaTareas;
ENDIF

 ELSEIF vec{i}=32 THEN

 IF vec{i-1}=10 OR vec{i-1}=11 OR vec{i-1}=12 OR vec{i-1}=20 OR vec{i-1}=21 OR vec{i-1}=22 THEN
 MoveL Apro_L_1_2, v300,fine,Servo_L;
 WaitSyncTask sync_RL_E, ListaTareas;
 Girar_B2_2;
 WaitSyncTask sync_RL_F, ListaTareas;
 WaitSyncTask sync_RL_G, ListaTareas;

ELSE
Agarre_L5_2;
WaitSyncTask sync_RL_E,ListaTareas;
ENDIF
```

Código brazo derecho

El código del brazo derecho es análogo al del izquierdo, a excepción de que es el encargado de tomar el cubo en la mesa y de dejarlo una vez ha sido resuelto. Por lo demás, los movimientos que realiza el brazo derecho son simétricos a los del izquierdo, coordinándose de forma que cuando uno tenga que girar una cara el otro sujete y posicione el cubo y viceversa.

Programa Principal

```
PROC Principal()
    Toma_Cubo_Mesa        !Toma del cubo de la mesa
    Toma_Fotos_R;          !Toma de las fotos y entrega del cubo al brazo izquierdo
    Recibir_Movimientos_R; !Conexión con Matlab y recepción de los movimientos
    Resolución_Cubo_R;     !Sincronización con brazo izquierdo y ejecución de los movimientos
    Dejar_Cubo_Mesa        !Depositar el cubo resuelto en la mesa
ENDPROC
```

Subprogramas

Toma_Cubo_Mesa

```
      SocketCreate server; !Creación del socket vinculado al servidor Server1
      SocketBind server, "192.168.125.1",14044; !Definición del puerto y la IP del socket
      SocketListen server; !Espera a la confirmación del cliente
      SocketAccept server,client; !Creación de la comunicación

  SocketReceive client1,\RawData:=data;
  UnpackRawBytes data,1,PosCuboY1\Ascii:=3;
  oki:=StrToVal(PosCuboX1,PosCuboX);

    SocketReceive client1,\RawData:=data;
  UnpackRawBytes data,1,PosCuboY1\Ascii:=3;
  oki:=StrToVal(PosCuboY1,PosCuboY);

     SocketReceive client1,\RawData:=data;
  UnpackRawBytes data,1,Orientacion1\Ascii:=2;
  oki:=StrToVal(Orientacion1,O1);

      SocketReceive client1,\RawData:=data;
UnpackRawBytes data,1,Orientacion2\Ascii:=2;
oki:=StrToVal(Orientacion2,O2);

      SocketReceive client1,\RawData:=data;
UnpackRawBytes data,1,Orientacion3\Ascii:=2;
oki:=StrToVal(Orientacion3,O3);

      SocketReceive client1,\RawData:=data;
UnpackRawBytes data,1,Orientacion4\Ascii:=2;
oki:=StrToVal(Orientacion4,O4);

 PosicionCubo:=[[PosCuboX,PosCuboY,PosCuboZ],[O1,O2,O3,O4],[1,0,0,0],[-150.651522324,9E+09,9E+09,9E+09,9E+09,9E+09]];
 PosicionCubo.trans:=[PosCuboX,PosCuboY,PosCuboZ];
 MoveJ PosicionCubo,v1000,z100,Servo\WObj:=WO_Cubo;

g_GripIn \holdForce:=10;
WaitTime 0.1;
GripLoad Cubo;
```

Toma_Fotos_R

```
WaitTime 3;
Foto1;
!enviar mensaje al cliente
SocketSend client1,\str:="A";
!recivir mensaje
SocketReceive client1,\RawData:=data;
UnpackRawBytes data,1,foto\Ascii:=2;
oki:=StrtoVal(foto,numfoto);
WaitUntil numfoto=2;

Foto2;
 !enviar mensaje al cliente
SocketSend client1,\str:="B";
!recivir mensaje
SocketReceive client1,\RawData:=data;
UnpackRawBytes data,1,foto\Ascii:=2;
oki:=StrtoVal(foto,numfoto);
WaitUntil numfoto=3;

Foto3;
!enviar mensaje al cliente
SocketSend client1,\str:="C";
!recivir mensaje
SocketReceive client1,\RawData:=data;
UnpackRawBytes data,1,foto\Ascii:=2;
oki:=StrtoVal(foto,numfoto);
WaitUntil numfoto=4;

 SocketSend client1,\str:="Z";

 !Cerrar comunicacion
 SocketClose server1;

 WaitSyncTask sync_RL_A, ListaTareas;
 WaitTime 12;
 Foto4;

 WaitSyncTask sync_RL_B, ListaTareas;
 WaitTime 3;
 Agarre_R5;
```

Recibir_Movimientos_R

```
        SocketCreate server1; !Creación del socket vinculado al servidor Server1
        SocketBind server1, "192.168.125.1",14044; !Definición del puerto y la IP del socket
        SocketListen server1; !Espera a la confirmación del cliente
        SocketAccept server1,client1; !Creación de la comunicación

   SocketReceive client1,\RawData:=data;
   UnpackRawBytes data,1,Recmov\Ascii:=2;
   oki:=StrToVal(Recmov,nummov);

   IF nummov<27 THEN
   mov:=nummov*3+1;
   SocketReceive client1,\RawData:=datos;
   UnpackRawBytes datos,1,Recmov1\Ascii:=mov;

    FOR j FROM 1 TO nummov DO
     part:=StrPart(Recmov1,B,2);
     ok:=StrToVal(part,vec{j});
     B:=B+3;
    ENDFOR
  ELSE
      SocketReceive client1,\RawData:=datos;
      UnpackRawBytes datos,1,Recmov1\Ascii:=79;

   FOR j FROM 1 TO 26 DO
    part:=StrPart(Recmov1,B,2);
    ok:=StrToVal(part,vec{j});
    B:=B+3;
   ENDFOR

   nummov:=nummov-26;
   mov:=nummov*3+1;

   SocketReceive client1,\RawData:=datos;
  UnpackRawBytes datos,1,Recmov1\Ascii:=mov;

  B:=2;
   FOR j FROM 1 TO nummov DO
    part:=StrPart(Recmov1,B,2);
    ok:=StrToVal(part,vec1{j});
    B:=B+3;
   ENDFOR
  ENDIF

       SocketClose server1;

           WaitSyncTask sync_RL_8, ListaTareas_RL;

    ENDPROC


ENDMODULE
```

Resolver_Cubo_R

```
FOR i FROM 1 TO nummov DO

    IF CuboTres = 0 THEN

    IF vec{i}=10 THEN
     MoveL Apro_R5,v1000,fine,Servo_R;
     WaitSyncTask sync_RL_E, ListaTareas;
     Girar_F;
     WaitSyncTask sync_RL_F, ListaTareas;
     WaitSyncTask sync_RL_G, ListaTareas;


     ELSEIF vec{i}=11 THEN
     MoveL Apro_R5,v1000,fine,Servo_R;
     WaitSyncTask sync_RL_E, ListaTareas;
     Girar_F1;
     WaitSyncTask sync_RL_F, ListaTareas;
     WaitSyncTask sync_RL_G, ListaTareas;

     ELSEIF vec{i}=12 THEN
     MoveL Apro_R5,v1000,fine,Servo_R;
     WaitSyncTask sync_RL_E, ListaTareas;
     Girar_F2;
     WaitSyncTask sync_RL_F, ListaTareas;
     WaitSyncTask sync_RL_G, ListaTareas;

     ELSEIF vec{i}=20 THEN
     WaitTime 3;
     Girar_R;
     WaitTime 10;
     WaitSyncTask sync_RL_E, ListaTareas;

     ELSEIF vec{i}=21 THEN
     WaitTime 3;
     Girar_R1;
     WaitSyncTask sync_RL_E, ListaTareas;

     ELSEIF vec{i}=22 THEN
     WaitTime 3;
     Girar_R2;
     WaitSyncTask sync_RL_E, ListaTareas;

     ELSEIF vec{i}=30 THEN
     WaitTime 3;
     Girar_B;
     WaitSyncTask sync_RL_E, ListaTareas;
```

```
ELSEIF vec{i}=31 THEN
WaitTime 3;
Girar_B1;
WaitSyncTask sync_RL_E, ListaTareas;

ELSEIF vec{i}=32 THEN
WaitTime 3;
Girar_B2;
WaitSyncTask sync_RL_E, ListaTareas;

ELSEIF vec{i}=40 THEN
Agarre_R5;
WaitSyncTask sync_RL_E, ListaTareas;

ELSEIF vec{i}=41 THEN
Agarre_R5;
WaitSyncTask sync_RL_E, ListaTareas;

ELSEIF vec{i}=42 THEN
Agarre_R5;
WaitSyncTask sync_RL_E, ListaTareas;

ELSEIF vec{i}=50 THEN
Agarre_R5;
WaitSyncTask sync_RL_E, ListaTareas;
ELSEIF vec{i}=51 THEN
Agarre_R5;
WaitSyncTask sync_RL_E, ListaTareas;

ELSEIF vec{i}=52 THEN
Agarre_R5;
WaitSyncTask sync_RL_E, ListaTareas;

ELSEIF vec{i}=60 THEN
 WaitTime 3;
 Rotar_R;
 WaitSyncTask sync_RL_E, ListaTareas;
 WaitSyncTask sync_RL_F, ListaTareas;
 WaitTime 3;
 Rotar_R1;
 WaitSyncTask sync_RL_G, ListaTareas;

ELSEIF vec{i}=61 THEN
 WaitTime 3;
 Rotar_R;
 WaitSyncTask sync_RL_F, ListaTareas;
 WaitTime 3;
 Rotar_R1;
 WaitSyncTask sync_RL_G, ListaTareas;
```

```
      ELSEIF vec{i}=62 THEN
       WaitTime 3;
       Rotar_R;
       WaitSyncTask sync_RL_E, ListaTareas;
       WaitSyncTask sync_RL_F, ListaTareas;
       WaitTime 3;
       Rotar_R1;
       WaitSyncTask sync_RL_G, ListaTareas;

  ENDIF
  ENDIF



!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!

      IF CuboTres = 1 THEN

      IF vec{i}=10 THEN
       MoveL Apro_R5,v1000,fine,Servo_R;
       WaitSyncTask sync_RL_E, ListaTareas;
       Girar_F_2;
       WaitSyncTask sync_RL_F, ListaTareas;
       WaitSyncTask sync_RL_G, ListaTareas;


       ELSEIF vec{i}=11 THEN
       MoveL Apro_R5,v1000,fine,Servo_R;
       WaitSyncTask sync_RL_E, ListaTareas;
       Girar_F1_2;
       WaitSyncTask sync_RL_F, ListaTareas;
       WaitSyncTask sync_RL_G, ListaTareas;

       ELSEIF vec{i}=12 THEN
       MoveL Apro_R5,v1000,fine,Servo_R;
       WaitSyncTask sync_RL_E, ListaTareas;
       Girar_F2_2;
       WaitSyncTask sync_RL_F, ListaTareas;
       WaitSyncTask sync_RL_G, ListaTareas;

        ELSEIF vec{i}=20 THEN
        WaitTime 3;
        Girar_R_2;
        WaitSyncTask sync_RL_E, ListaTareas;

        ELSEIF vec{i}=21 THEN
        WaitTime 3;
        Girar_R1_2;
        WaitSyncTask sync_RL_E, ListaTareas;

        ELSEIF vec{i}=22 THEN
        WaitTime 3;
        Girar_R2_2;
        WaitSyncTask sync_RL_E, ListaTareas;
```

ç

```
ELSEIF vec{i}=30 THEN
IF vec{i-1}=10 OR vec{i-1}=11 OR vec{i-1}=12 OR vec{i-1}=20 OR vec{i-1}=21 OR vec{i-1}=22 THEN
Agarre_R5_2;
WaitSyncTask sync_RL_E, ListaTareas;

ELSE
WaitTime 3;
Girar_B_2;
WaitSyncTask sync_RL_E, ListaTareas;
ENDIF

ELSEIF vec{i}=31 THEN
IF vec{i-1}=10 OR vec{i-1}=11 OR vec{i-1}=12 OR vec{i-1}=20 OR vec{i-1}=21 OR vec{i-1}=22 THEN
Agarre_R5_2;
WaitSyncTask sync_RL_E, ListaTareas;

ELSE
WaitTime 3;
Girar_B1_2;
WaitSyncTask sync_RL_E, ListaTareas;
ENDIF
ELSEIF vec{i}=32 THEN
IF vec{i-1}=10 OR vec{i-1}=11 OR vec{i-1}=12 OR vec{i-1}=20 OR vec{i-1}=21 OR vec{i-1}=22 THEN
Agarre_R5_2;
WaitSyncTask sync_RL_E, ListaTareas;

ELSE
WaitTime 3;
Girar_B2_2;
WaitSyncTask sync_RL_E, ListaTareas;
ENDIF
```

```
  ELSEIF vec{i}=40 THEN
  Agarre_R5_2;
  WaitSyncTask sync_RL_E, ListaTareas;

  ELSEIF vec{i}=41 THEN
  Agarre_R5_2;
  WaitSyncTask sync_RL_E, ListaTareas;

  ELSEIF vec{i}=42 THEN
  Agarre_R5_2;
  WaitSyncTask sync_RL_E, ListaTareas;

  ELSEIF vec{i}=50 THEN
  Agarre_R5_2;
  WaitSyncTask sync_RL_E, ListaTareas;

  ELSEIF vec{i}=51 THEN
  Agarre_R5_2;
  WaitSyncTask sync_RL_E, ListaTareas;

  ELSEIF vec{i}=52 THEN
  Agarre_R5_2;
  WaitSyncTask sync_RL_E, ListaTareas;
```

```
ELSEIF vec{i}=60 THEN
IF vec{i-1}=40 OR vec{i-1}=41 OR vec{i-1}=42 OR vec{i-1}=50 OR vec{i-1}=51 OR vec{i-1}=52 THEN
WaitTime 3;
Girar_D_2;
WaitSyncTask sync_RL_E, ListaTareas;

ELSE
 WaitTime 3;
 Rotar_R_2;
 WaitSyncTask sync_RL_E, ListaTareas;
 WaitSyncTask sync_RL_F, ListaTareas;
 WaitTime 3;
 Rotar_R1_2;
 WaitSyncTask sync_RL_G, ListaTareas;

ELSEIF vec{i}=61 THEN
IF vec{i-1}=40 OR vec{i-1}=41 OR vec{i-1}=42 OR vec{i-1}=50 OR vec{i-1}=51 OR vec{i-1}=52 THEN
WaitTime 3;
Girar_D1_2;
WaitSyncTask sync_RL_E, ListaTareas;

ELSE
 WaitTime 3;
 Rotar_R_2;
 WaitSyncTask sync_RL_E, ListaTareas;
 WaitSyncTask sync_RL_F, ListaTareas;
 WaitTime 3;
 Rotar_R1_2;
 WaitSyncTask sync_RL_G, ListaTareas;

ELSEIF vec{i}=62 THEN
IF vec{i-1}=40 OR vec{i-1}=41 OR vec{i-1}=42 OR vec{i-1}=50 OR vec{i-1}=51 OR vec{i-1}=52 THEN
WaitTime 3;
Girar_D2_2;
WaitSyncTask sync_RL_E, ListaTareas;

ELSE
 WaitTime 3;
 Rotar_R_2;
 WaitSyncTask sync_RL_E, ListaTareas;
 WaitSyncTask sync_RL_F, ListaTareas;
 WaitTime 3;
 Rotar_R1_2;
 WaitSyncTask sync_RL_G, ListaTareas;
```

Dejar_Cubo_Mesa

```
Agarre_R5;
waittime 0.1;
MoveJ Apro_DejarCubo, v300, fine, servo_R;
MoveL DejarCubo, v300, fine, servo_R;
g_GripOut;
waittime 0.1;
MoveL Apro_DejarCubo, v300, fine, servo_R;
MoveJ Agarre_R, v300, fine, servo_R;
```

ANEXO III. HOJAS DE ESPECIFICACIONES

ROBOT YUMI IRB 14000

ROBOTICS

YuMi® creating an automated future together.

You and me.

YuMi is the first truly collaborative dual armed robot, designed for a world in which humans and robots work together. It heralds a new era of robotic co-workers which are able to work side-by-side on the same tasks as humans with extreme accuracy while ensuring the safety of those around it.

Collaboration
YuMi is designed to meet the flexible and agile production needs required for small parts assembly in the electronics industry. It is also well suited to other small parts environments, including the manufacture of watches, toys and automotive components. All of this thanks to its dual-arms, flexible hands, universal parts feeding system, camera-based part location and state-of-the-art motion control.

Redefining safety
YuMi has a lightweight yet rigid magnesium skeleton covered with a floating plastic casing wrapped in soft padding, which absorbs the force of any unexpected impacts to a very high degree. YuMi has no pinch points so that sensitive ancillary parts cannot be crushed between two opposing surfaces as the axes open and close.

If YuMi senses an unexpected impact or change in its environment such as a collision with a coworker, it can pause its motion within milliseconds to prevent injury, and the motion can be restarted again as easily as pressing play on a remote control.

YuMi is very precise and fast, returning to the same point in space over and over again to within 0.02 mm accuracy and moving at a maximum velocity of 1,500 mm/sec. This ensures the safety of human co-workers on production lines and in fabricating cells.

Total solution concept
ABB also develops software and manufactures hardware, peripheral equipment, process equipment and modular manufacturing cells. This "total solution" concept is evident in YuMi's breakthrough design.

Features
- The fifth-generation, integrated IRC5 controller with TrueMove and QuickMove™ motion control technology commands accuracy, speed, cycle-time, programmability and synchronization with external devices.
- I/O interfaces include Ethernet IP, Profibus, USB ports, DeviceNet™, communication port, emergency stop and air-to-hands. YuMi accepts a wide range of HMI devices including ABB's teach pendant, industrial displays and commercially available tablets.
- The 100-240 volt power supply plugs into any power socket for worldwide versatility.

Benefits
- Can operate equally effectively side-by-side or face-to-face with human coworkers.
- Servo grippers (the "hands") include options for built-in cameras.
- Real-time algorithms set a collision-free path for each arm customized for the required task.
- Padding protects coworkers in high-risk areas by absorbing force if contact is made.

Specification

Robot version	Reach (mm)	Payload (g)	Armload
IRB 14000-0.5/0.5	559	500	No armloads
Number of axes	14		
Protection	Std: IP30 and Clean Room		
Mounting	Table		
Controller	Integrated		
Integrated signal and power supply	24V Ethernet or 4 Signals		
Integrated air supply	1 per Arm on tool Flange (4 Bar)		
Integrated Ethernet	One 100/10 Base-TX ethernet port/per arm		

Performance (according to ISO 9283)

IRB 14000-0.5/0.5	
0.5 kg picking cycle	
25* 300 * 25 mm	0.86s
Max TCP Velocity	1.5 m/s
Max TCP Acceleration	11 m/s*s
Acceleration time 0-1m/s	0.12s
Position repeatability	0.02 mm

Technical information

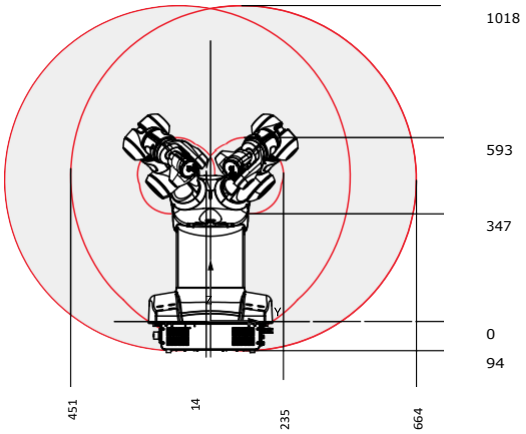
Physical	
Robot base	399 x 496 mm
Robot toes	399 x 134 mm
Weight	38 kg
Environment	
Ambient temperature for mechanical unit	
During operation	+5°C i (41°F) to +40°C (104°F)
During transportation and storage	-10°C (14°F) to +55°C (131°F)
Relative humidity	Max. 85%
Noise level	< 70 dB
Safety	PL b Cat B

Data and dimensions may be changed without notice.

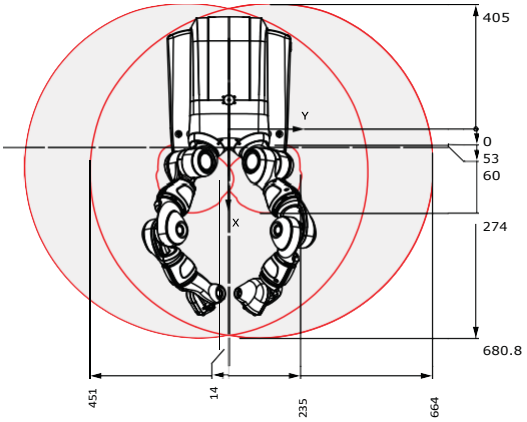
Movement

Axis movement	Working range	Axis max. speed
Axis 1 arm rotation	-168.5° to +168.5°	180°/s
Axis 2 arm bend	-143.5° to +43.5°	180°/s
Axis 7 arm rotation	-168.5° to +168.5°	180°/s
Axis 3 arm bend	-123.5° to +80°	180°/s
Axis 4 wrist rotation	-290° to +290°	400°/s
Axis 5 wrist bend	-88° to +138°	400°/s
Axis 6 flange rotation	-229° to +229°	400°/s

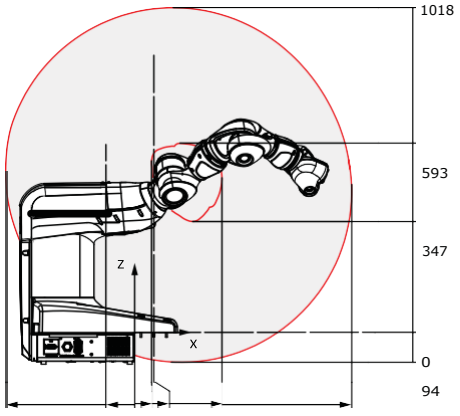
Working range, front view



Working range, top view

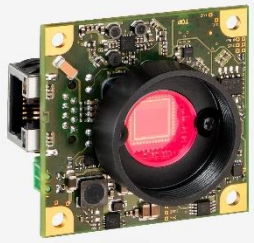


Working range, side view



CÁMARA UI 5584LE-C-HQ

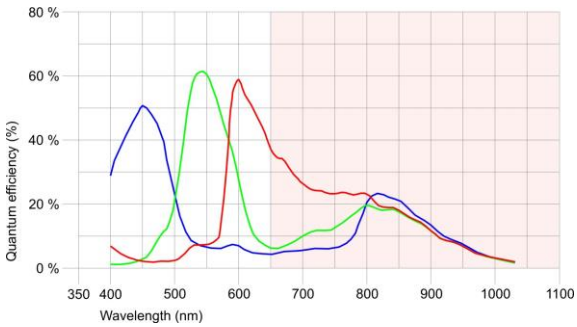
UI-5584LE-C-HQ (AB00469)



Made in Germany

Especificación

Tipo de sensor	CMOS Color
Sistema de obturador	Rolling Shutter / Global Start Shutter
Characteristic	Lineal
Método de lectura del sensor	Progressive scan
Clase de píxeles	QSXGA
Resolución	4,92 Mpx
Resolución (h x v)	2560 x 1920 Pixel
Relación de aspecto	4:3
CAD	12 bit
Profundidad de color (camára)	12 bit
Clase de sensor óptico	1/2"
Superficie optica	5,632 mm x 4,224 mm
Diagonal del sensor óptico	7,04 mm (1/2,27")
Tamaño de píxel	2,2 µm
Fabricante	ON Semiconductor
Denominación del sensor	MT9P006STC
Ganancia (total/RGB)	12.2x/5.8x
AOI horizontal	mayor frecuencia de imagen
AOI vertical	mayor frecuencia de imagen
AOI ancho de imagen / ancho de paso	32 / 4
AOI alto de imagen / ancho de paso	4 / 2
AOI cuadrícula de posición (horizontal/vertical)	4 / 2
Binning horizontal	mayor frecuencia de imagen
Binning vertical	mayor frecuencia de imagen
Método binning	Color
Factor binning	2 / 3 / 4 / 6
Subsampling horizontal	mayor frecuencia de imagen



Modelo

Rango de frecuencia de píxeles	4 MHz - 96 MHz
Frecuencia de imagen en modo libre	14
Frecuencia de imágenes disparador (máxima)	14
Tiempo de exposición (mínimo - máximo)	0.034 ms - 3404 ms
Consumo de potencia	2,6 W - 3,1 W
Memoria gráfica	60 MB

Condiciones ambientales

Las temperaturas mencionadas describen la temperatura del aparato exterior de la carcasa de la cámara.

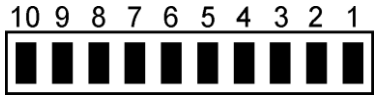
Temperatura del aparato durante el funcionamiento	0 °C - 55 °C / 32 °F - 131 °F
Temperatura del aparato durante el almacenamiento	-20 °C - 60 °C / -4 °F - 140 °F
Humedad (relativa, sin condensación)	20 % - 80 %

Conexiones

Conexión de interfaz	GigE RJ45
Conexión I/O	Conector de 10 polos (Pico Blade) Molex
Alimentación	12 V - 24 V

Asignación de pins conexión I/O

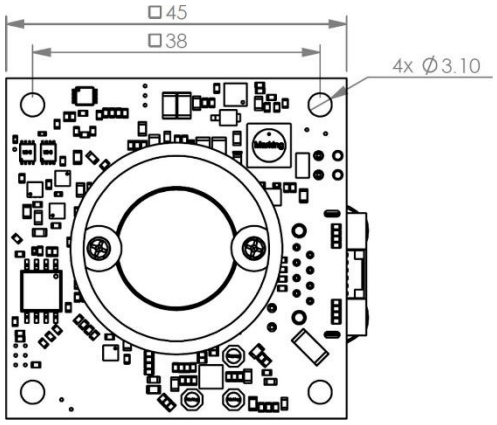
1	Masa (GND)
2	Vout 3,1 V máx. 100 mA
3	Entrada de disparador sin optoacoplador
4	Salida de flash sin optoacoplador
5	General Purpose I/O (GPIO) 1
6	General Purpose I/O (GPIO) 2
7	Señal de reloj bus I2C
8	Señal de datos bus I2C
9	Vin+ 12 V (160 mA) - 24 V (90 mA)

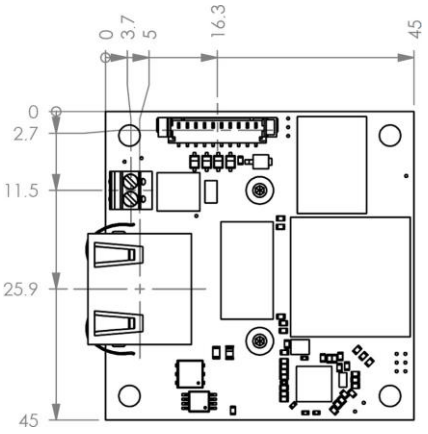


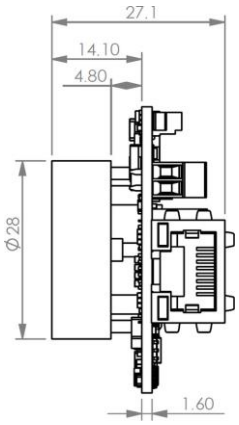
Vista de la cámara

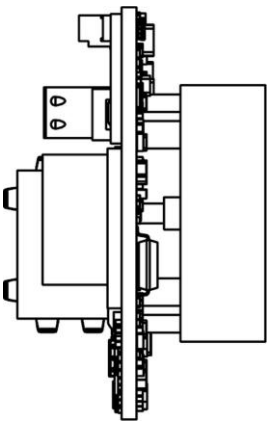
Diseño

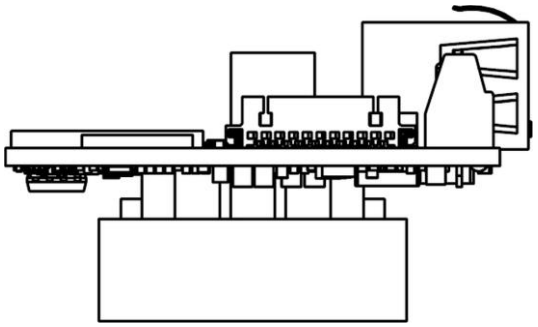
Conexión del objetivo	Montura CS / Montura C
Grado de protección IP	-
Dimensiones	45,0 mm x 45,0 mm x 27,1 mm
Peso	24 g


□45
□38
4x Ø3.10


0
3.7
5
16.3
45
0
2.7
11.5
25.9
45


27.1
14.10
4.80
Ø28
1.60





DOMO DE LUCES LED

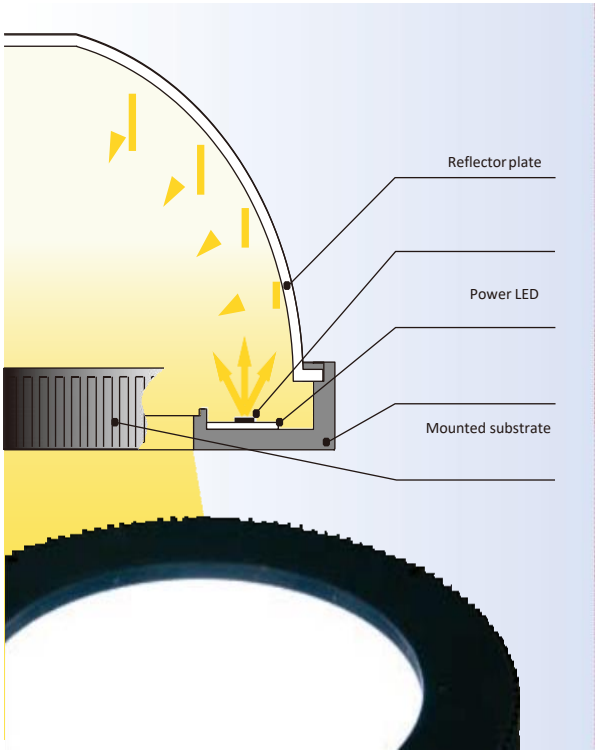
Dome Lights
HPD Series

"Brighter" "More uniform" "Easy to use" High-Power Dome Lights

Enhanced light intensity and larger uniform area allows for use in more diversified applications.

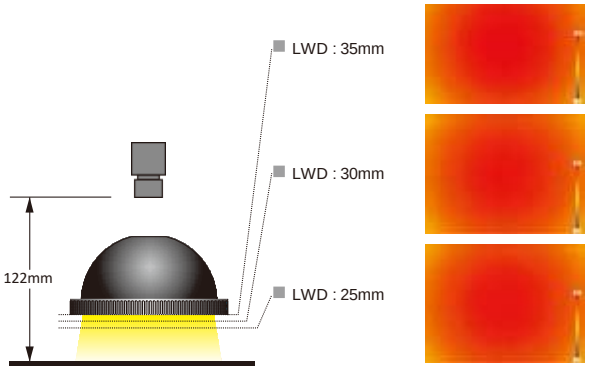
The use of power LEDs greatly enhances the light intensity compared with conventional dome lights. The unique illumination structure allows for a larger uniform area.

Achievement of high-luminosity uniform diffusion light



Enhanced light intensity and larger uniform area broaden the use of the lights to more diverse environments and applications.

Achievement of larger uniform areas



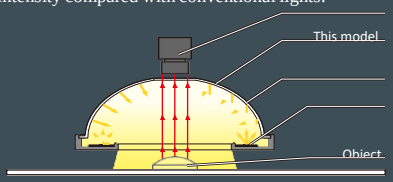
* This shows the relative brightness distribution when the brightest area is set at 100.

The data given here is intended for reference purposes only and is not intended to assure the quality of the product.

Measuring conditions	Camera	1/2 inch CCD
	Lens	f12mm
	Macro ring	0mm
	WD	122mm
	Field (Y direction)	50mm
	Lighting	HPD-150SW
	LWD	25mm / 30mm / 35mm

Illumination structure of HPD-150

The use of power LEDs significantly enhances the light intensity compared with conventional lights.



Examples of surface-emitting dome light images

Macroscopic image

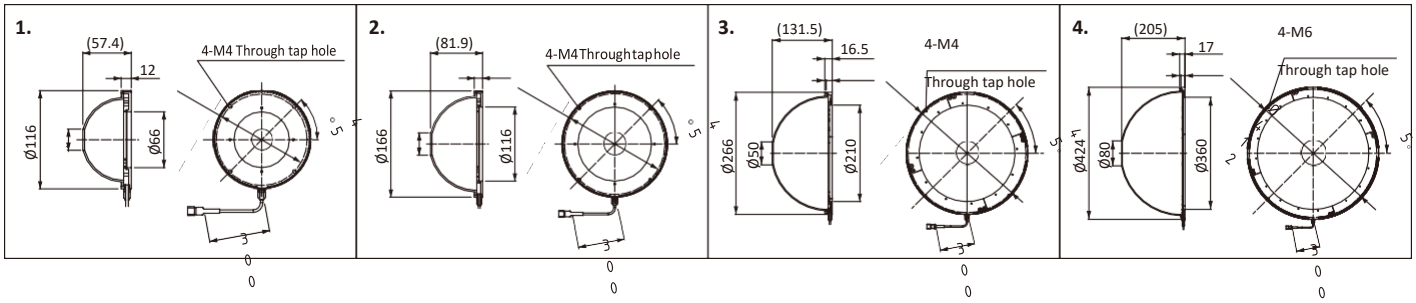
Unevenness occurs in the illumination area with the 100W halogen ring lamp.

The surface of a workpiece is imaged evenly and brightly with the high-power dome light.

Series	Direct Number	Model Name	Color	Power Consumption	Option	Dimension
HPD	1002935	HPD-100RD	●	24V / 9.0W	—	1
	1002934	HPD-100SW	○	24V/14W		
	1002936	HPD-100BL	●			
	1002941	HPD-150RD	●	24V / 16W	—	2
	1002940	HPD-150SW	○	24V/20W		
	1002942	HPD-150BL	●			
	1003214	HPD-250RD	●	24V / 25W	—	3
	1003213	HPD-250SW	○	24V/37W		
	1003215	HPD-250BL	●			
	1003217	HPD-400RD	●	24V / 25W	—	4
	1003216	HPD-400SW	○	24V/41W		
	1003218	HPD-400BL	●			

*HPD Series cannot be used in combination with CCS Strobe Control Unit (PTU2 Series, etc.)
*The peak wavelength for Red lights is 625 nm. If a sharp-cut filter is required, use a R60 Filter (optional).
*For further details on these options, refer to page 103.

Dimension Diagrams (Unit: mm)



ESPAÑA
BARCELONA
Vergós, 55
08017 Barcelona
Tel. 93- 252 57 57
Fax.93- 252 57 58

MADRID
Ribera de Loira, 46
28042 Madrid
Tel.- 902 46 32 46
Fax.- 91 503 00 99

infaimon@infaimon.com

PORTUGAL
Rua de Viseu, 43
3800-280 Aveiro
Tel +351- 234 31 2034
Fax +351- 234 31 2035
infaimon.pt@infaimon.com

MÉXICO
Hacienda Chintepec nº 110.
Col. El Jacal.
76180 Querétaro. QRO.
Tel. +52 442 215 14 15
Fax. +52 442 215 35 15
infaimon.mx@infaimon.com

BRASIL
Dr Heitor Nascimento, 196, Bl A - Sl 75
Cond. Aliança - Bairro Morumbi
13140-695 Paulínia - SP
Tel. +55 19 2513 0450
Fax. +55 19 2513 0231
infaimon.br@infaimon.com

www.infaimon.com